НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проект

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Система тестування на основі відносного оцінювання по статистичним вибіркам»

Виконав:
студент IV курсу, групи ІА-62
Коваленко І. В. _____

Керівник:
Асистент,
Вовк Євгеній Андрійович _____

Рецензент:
Доктор, к. т. н
Сперкач Майя Олегівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Коваленко Ігор Валерійович

1. Тема проєкту «Система тестування на основі відносного оцінювання по статистичним вибірками», керівник проєкту Вовк Євгеній Андрійович, асистент, затверджені наказом по університету від «_7_»_травня_ 2020 р. №_1081_

2. Термін подання студентом проєкту 09.06.2020

3. Вихідні дані до проєкту: система тестування на основі відносного оцінювання по статистичним вибіркам.

4. Зміст пояснювальної записки: 4.1 Аналіз предметної області; 4.2 Пошук технічного рішення; 4.3 Власна реалізація.

5. Перелік графічного матеріалу: 5.1 ER-діаграма бази даних; 5.2 Структура клієнтської частини додатку; 5.3 Компонентна структура сторінки авторизації; 5.4 Діаграма послідовності; 5.5 Діаграма основних варіантів використання системи.

6. Дата видачі завдання 05.03.2020

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз предметної області	04.04.2020	
2	Огляд існуючих рішень	10.04.2020	
3	Пошук архітектурного рішення	13.04.2020	
4	Пошук технічного рішення	19.04.2020	
5	Проектування бази даних	26.04.2020	
6	Проектування клієнтської частини та міжсерверної взаємодії	02.05.2020	
7	Проектування графічного інтерфейсу користувача	17.05.2020	
8	Реалізація та відлагодження індивідувальної частини комплексного дипломного проєкту	30.05.2020	
9	Оформлення текстової та графічної документації	05.06.2020	

Студент Ігор Коваленко

Керівник Євгеній Вовк

АНОТАЦІЯ

У дипломному проекті розглянуто концепцію системи тестування, проведено огляд наявних на ринку рішень. Сформовані вимоги до сучасної тестової системи. Запропоновано удосконалення існуючих рішень шляхом введення динамічного показника ваги правильної відповіді, використання сучасних технологій та підходів. Проведено аналіз наявних технології збереження даних, розробки клієнтської частини веб-орієнтованих систем. Розглянуто основні архітектурні підходи для побудови веб-орієнтованих систем. На основі цього розроблено та детально описано індивідуальну частину системи тестування у межах комплексного дипломного проекту.

Ключові слова: система тестування, веб-додаток.

Розмір пояснювальної записки 65 – аркушів, 12 ілюстрацій й 5 додатків.

ANNOTATION

In the diploma project, the concept of the testing system is considered, the review of the decisions available in the market is carried out. Requirements to the modern test system are formed. It is proposed to improve existing solutions by introducing a dynamic weight index of the correct answer, the use of modern technologies, and approaches. The analysis of available data storage technologies, development of the client part of web-oriented systems is carried out. The main architectural approaches for building web-oriented systems are considered. Based on this, an individual part of the testing system was developed within the bounds of a complex diploma project.

Keywords: testing system, web application.

The size of the explanatory note is 65 - sheets, 12 illustrations, and 5 applications.

Пояснювальна записка
до дипломного проекту
на тему: «Система тестування на основі відносного оцінювання по статистичним вибіркам»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

ACID – Atomicity, Consistency, Isolation, Durability

API – application programming interface

ASP – Active Server Pages

BASE – Basic Availability, Soft state, Eventual consistency

DNS – Domain Name System

DOM – Document Object Model

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

MVC – Model-View-Controller

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

SSG – static site generation

TTI – time-to-interactive

XML – eXtensible Markup Language

БД – база даних

ІТ – інформаційні технології

ОС – операційна система

ПЗ – програмне забезпечення

СУБД – система управління базами даних

СУРБД - система управління реляційними базами даних

ВСТУП

У сучасних реаліях ні одна сфера життя не обходиться без оцінки якості рівня знань та підготовки спеціалістів по тим чи іншим напрямам. У результаті постає завдання якісної та коректної оцінки знань з можливістю їх подальшого вдосконалення в межах процесу навчання. Воно може бути представлене процесом виявлення відсутності знань, в межах певного графу знання по тій чи іншій спеціальності.

Виходячи з цього, створюються безліч платформ, програм та утиліт, які частково вирішують завдання, проте розглядаючи існуючі рішення та наявні життєві приклади можна говорити про їх недосконаліть, неточність та нешаблонність.

З розвитком технологій, все більше функцій класичних додатків переносяться на веб-платформи, втілюючи функціонал у вигляді веб-сервісів. Причиною цьому є зручність створення інтерфейсів на базі мови HTML та відповідних препроцесорів і абсолютна кросплатформеність даних рішень.

Даний тренд не оминув і системи тестування. Сьогодні більшість з них представлена в мережі інтернет на відповідних ресурсах.

Зі збільшенням обсягу функціоналу представленого на веб-ресурсах, галузь розвивалася і в результаті майже всі проблеми розгортання сервісів на базі веб-платформ були вирішені. Останніми кроками стали вирішення проблем перезавантаження сторінок, роботи в офлайн-режимі, імплементації складної графіки та комплексних інтерфейсів з широким функціоналом. Це стало можливим завдяки розповсюдженню технологій service-workers та SPA. З використанням SPA, зникають обмеження в імплементації складної логіки і з’являється можливість створювати повноцінні клієнтські додатки на базі мов HTML та JavaScript.

Більшість наявних сьогодні систем тестування не об’єднують тестування та навчання шляхом встановлення незнання та його ліквідацію за допомогою опису і вказання правильних відповідей шляхом пояснень. Крім

того ні одна з них не вирішує проблему еквівалентності ваги правильних відповідей. Також присутня проблема різноманітності форми тестових завдань і сучасності UX частини систем. Як приклад Moodle, мережева академія Cisco мають застарілий дизайн та вузький перелік форм тестових питань. Консервативність у використанні сучасних веб-технологій значно обмежує їх функціонал і актуальність.

В результаті метою даної роботи є розробка клієнтської частини та бази даних системи тестування на основі відносного оцінювання по статистичним вибіркам. Дана система повинна об’єднати розроблене рішення проблеми якості оцінювання та сучасні технології у сфері веб-застосунків для створення нового, унікального рішення.

Результати виконаної дослідницької роботи по даній темі були опубліковані у матеріалах конференції «Система тестування на основі відносного оцінювання по статистичним вибіркам» на конференції «Summer Infocom Anvanced Solutions» в 2020 році в м. Києві.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення тесту, принципи формування тестових завдань

Тест – це завдання стандартної форми, призначене для виявлення наявності, або відсутності знань, навичок та вмінь, або інших когнітивних характеристик. [1]

Уцілому, тести можна поділити на тести успішності (знань, вмінь, тощо), здібностей і особистісні. У даній роботі сфокусовано увагу на питанні створення тестів успішності та здібностей, проте не виключається використання розробленої системи для особистісного тестування.

Для тестів можна виділити наступні характерні особливості:

- простота процедури тестування;
- фіксація результатів відбувається безпосередньо;
- варіативність індивідуального і групового підходів до використання;
- консиситентність математичних оцінок;
- короткочасність;
- нормованість.

Основні принципи формування тестових звадань: [1]

- узгодженість засобів та цілей. Тестові завдання та принципи їх формування мають відповідати цілям та принципам формування системи навчального процесу уцілому. Тому перед початком розробки тестових звадать має бути зроблений і врахований огляд мети та засобів проведення тестування;
- врахування цілей вивчення дисципліни. Цілі вивчення дисциапліни можна поділити на внутрішні, зовнішні, тактичні, стратегічні і оперативні. Враховується відповідна ієрархія цілей як для дисципліни в цілому, так і для складових розділів і модулів;
- здійснення тестування за відносно самостійною частиною навчальної дисципліни. Виправданість тестування визначається, метою оцінити рівень

засвоєння знань та вмінь за відповідною темою або розділом дисципліни, а не за матеріалом лекції;

- визначення ступеня досягнення поставлених цілей. Враховуючи визначення навчання, як елемент освіти, при вивченні дисципліни завжди ставляться наступні типи цілей: загальноосвітні, загальнонаукові, спеціальні тощо;
- відповідність психолого-фізіологічним властивостям пам'яті.

1.2 Огляд ринку та класифікація

З розвитком систем навчання, з’явилася необхідність розробки сучасних систем тестування. Згодом, вони стали потрібною та важливою складовою сучасного навчального процесу. [2]

На сьогоднішній день існує безліч систем онлайн-тестування. [3] Скориставшись пошуковою системою Google, та переглянувши знайдені веб-ресурси і рейтинги, можна знайти ряд схожих за функціоналом та призначенням систем.

Відповідно до призначення тести поділяються на: [4]

- навчальні – ті, що допомагають закріпити вивчений матеріал. Зазвичай такі тести можна знайти після кожної глави у курсі в якості невеликої практики. Відсутні обмеження по часу, штрафи за неправильні відповіді. На вирішення задач надається кілька спроб. Після кожної помилки відображаються пояснення. Як приклад, тести що стають доступні після завершення вивчення глав у системі мережевої академії Cisco;
- атестаційні – допомагають оцінити знання користувача. В таких тестах присутні обмеження по часу, дається одна спроба на відповідь, відсутні пояснення помилок. Як приклад, Exam.net.

За спеціалізацією, тестові системи можна поділити на: [4]

- спеціальні – ті, що призначені для застосування лише у певних сферах. Як приклад, нині популярні системи тестування персоналу SHL, TalentQ, Kenexa, або система тестування інтегрована в платформу Udemy, призначена для оцінки засвоєних навичок після проходження курсу;
- загальні – ті, що мають широке призначення. Різноманітні конструктори тестів та більш загальні системи як Google Forms.

Більшість наявних систем не об’єднують тестування та навчання шляхом встановлення незнання та його ліквідацію за допомогою опису і вказання правильних відповідей шляхом пояснень. Крім того ні одна з них не вирішує проблему еквівалентності ваги правильних відповідей. Також присутня проблема різноманітності форми тестових завдань і сучасності UX частини систем. Як приклад Moodle, мережева академія Cisco мають застарілий дизайн та вузький перелік форм тестових питань.

1.3 Функціональні типи питань що підлягають реалізації в системі

Після огляду популярних систем тестування представлених на веб-порталах, зокрема мережевої академії Cisco, Canvas, Blackboard Learn, Kahoot! та на основі досвіду проходження тестів під час навчання, було сформовано перелік основних типів тестів. Для повної реалізації функціональних вимог і конкурентоздатності системи з переліку було обрано як ті, що підлягають імплементації, наступні види тестів:

- правильно/неправильно – користувач повинен визначити чи правдиве твердження в питанні. Це найпростіший тип питання;
- вибір однієї відповіді – користувачу потрібно вибрати одну правильну відповідь з запропонованих варіантів;
- вибір кількох відповідей – користувач обирає правильні варіанти зі списку. Завдання такого типу складніші чим вибір однієї відповіді, так як

кількість правильних відповідей заздалегідь невідома. Правильна відповідь методом випадкового підбору малоймовірна;

- коротка відповідь – користувач повинен ввести правильну відповідь в текстове поле. Для правильної відповіді необхідно добре розбиратись у темі питання;
- послідовність – користувач розміщує елементи у правильній послідовності;
- числова відповідь – користувач вводить число у поле для відповіді. Вгадати правильну відповідь малоймовірно;
- відповідність – користувач повинен з'єднати пари слів, фраз, або зображень. Додаткові «зайві» варіанти відповідності можуть ускладнити питання.

1.4 Проблема еквівалентності ваги правильних відповідей

Як правило, більшість тестових систем використовують наступний підхід до оцанювання відповідей. Студент відповідає на питання тесту, правильні відповіді зараховіються як один бал для простих питань, та як один, або більше для комплексних. Неправильні відповіді не зараховуються. В результаті сума зарахованих балів у відношенні до максимально можливих для зарахування сприймається як фінальна оцінка. Даний підхід є простим та зрозумалим. Отриманий бал може бути приведений до 12 або 100-бальної шкали за допомогою простої пропорції.

Проте підхід має ряд недоліків. При створенні тесту завжди є необхідність виставлення прохідного балу. Оскільки відсутній універсального рецепту для фіксованого значення прохідного балу. Крім того існує проблема еквівалентності ваги правильних відповідей, оскільки різні питання можуть посилатися на різні об'єми знань, можуть мати неточності, бути складними для розуміння.

Огляд наявних рішень показав, що проблема лишилася без уваги авторів систем даного класу. Тому необхідно розробити та реалізувати логіку відносного показника. Як рішення для забезпечення об’єктивності тестування прийнято до розробки підхід динамічного визначення ваги правильних відповідей відносно загальної статистики відповідей на питання тесту.

1.5 Формування технічного завдання та основні вимоги до розроблюваного рішення

Розроблювана система повинна, за можливістю, включати переваги існуючих рішень та не мати відповідних недоліків, бути розробленою у відповідності з основними принципами ООП та з використанням сучасних шаблонів програмування, мати систему ідентифікації користувача, збирати інформацію про проходження тестів користувачами та на їх основі корегувати фінальний бал, мати зручний інтерфейс користувача, який відповідає сучасним вимогам UX.

В результаті огляду прийнято наступні вимоги до проектованого рішення:

- можливість додавання пояснень та посилань до відповідних питанням тем та їх пов’язування;
- вирішення проблеми еквівалентності ваги правильних відповідей шляхом впровадження динамічного показника відносного загальній статистиці відповідей на питання тесту;
- впровадження різноманітних форм тестових питань з можливістю конфігурації;
- розробка сучасного дизайну;
- широка спеціалізація системи, шляхом налаштувань формату тестів.

Основним завданням розроблюваної системи є об’єктивна оцінка рівня знань людини по тій, чи іншій темі. Також завданням системи є виявлення

окремих областей знань, в яких людина не компетентна та надання довідкової інформації для ліквідування знайдених пробілів.

Для реалізації даного завдання система повинна реалізовувати наступні функції:

- проходження тесту, зокрема, відповіді на питання;
- статистичної оцінки ваги правильної відповіді;
- виявлення незнання та забезпечення довідки;
- контролю доступу до системи (автентифікації, авторизаціїї та реєстрації) ;
- роль викладача (автора тестів), яка реалізує логіку перегляду результатів тестів користувачів, редагування параметрі тесту, створення нових тестів;
- роль студента, яка реалізує логіку перегляду тестів, власних результатів, проходження нових;
- збереження даних.

1.6 Додаткові вимоги до системи

Для того, щоб охопити якнайбільшу область знань, але при цьому не виснажувати користувача, понижуючи його продуктивність було вирішено визначити задовільною довжину тесту 25-30 питань, з можливістю варіації. При цьому загальний банк питань повинен бути у 3-4 рази більшим ніж кількість питань у згенерованому тесті для забезпечення унікальності тестів у різних користувачів і виключити можливість плагіату та розповсюдження набору готових питань-відповідей на тест. Також для виключення можливості користування додатковими матеріалами користувачем необхідно, щоб тест передбачав обмеження по часу.

Окрім підбору рекомендованих параметрів, важливо зберегти можливість вільного вибору віповідних характеристик тесту автором, не обмежуючи його у виборі часу, кількості, складності питань, тощо. Оскільки

спектр можливих варіантів використання тестового функціоналу є додтатньо широким, у чому можна пересвідчитися з наявного огляду існуючих рішень, комбінація і може потребувати окремих налашувань.

1.7 Вимоги до сховища даних

Для імплементації функції збереження даних необхідно формалізувати перелік інформації, що підлягає збереженню та її логічні зв’язки між абстрактними сутностями що представляють окремі поняття, які підлягають збереженню. У відповідності з цим було сформовано наступні тези:

- у відповідності до функції проходження тесту, можна виділити наступні сутності які мають бути збережені системою: по-перше, власне питання в тесті; по-друге, можливі варіанти відповіді на дане питання; по-третє інформація про формат питання. Даний перелік сутностей формує цільне поняття «питання тесту». Тест складається з переліку відповідних питань. Також він повинен включати таку додаткову інформацію, як автора, список допущених студентів, тощо;
- для реалізації функції статистичної оцінки ваги правильної відповіді необхідно зберігати всі відповіді на окремі питання в рамках тесту, для того, щоб у подальшому, грунчуючись на їх правильності формувати вагові коефіцієнти;
- для виявлення окремих областей незнання користувача, що проходить тестування, необхідно зберігати інформацію про відповідність тем та довідкових матеріалів з тієї чи іншої області знань конкретному питанню тесту. Окрім цього це дозволить, у випадку неправильної відповіді користувача, сформувати набір інформації, який допоможе останньому ліквідувати пробіл знань;
- також необхідне збереження інформації для подальшої індентифікації користувача та реалізації функціоналу контролю доступу;

- для реалізації функціоналу викладача постає питання збереження списку створених тестів окремим викладачем, студентів допущених до проходження тесту;

- для реалізації функціоналу ролі студента необхідно зберігати інформацію про історію проходження користувачем тестів, додаткову інформацію про області незнань користувача, власне інформаццію користувача.

Перелічені тези формують вимоги до збереження даних. Сховище даних системи повинне зберігати відповідні взаємозв’язки між інформацією та власне інформацію, виключаючи можливості аномалій та пошкодження даних.

2 ПОШУК ТЕХНІЧНОГО РІШЕННЯ

2.1 Архітектурне рішення

2.1.1 Поняття архітектури програмного забезпечення

Архітектура програмного забезпечення [5] - це процес перетворення таких характеристик програмного забезпечення, як: гнучкість, масштабованість, можливість реалізації, багаторазовість використання і безпека в структуроване рішення, яке відповідає технічним і функціональним вимогам.

Архітектура ПЗ служить основою для системи. Вона забезпечує абстракцію для управління складністю системи та встановлює механізми зв'язку та координації між компонентами.

Вона визначає структуроване рішення для задоволення всіх технічних та експлуатаційних вимог, оптимізуючи загальні атрибути якості, такі як продуктивність та безпеку.

Крім того, вона передбачає набір значущих рішень щодо організації, пов'язаних з розробкою програмного забезпечення, і кожне з цих рішень може мати значний вплив на якість, підтримуваність, продуктивність та загальний успіх кінцевого продукту. Ці рішення включають:

- вибір структурних елементів та їх інтерфейсів, за якими складається система;
- поведінка, як зазначено у співпраці між цими елементами;
- складання цих структурних та поведінкових елементів у велику підсистему;
- архітектурні рішення узгоджуються з цілями бізнесу;
- архітектурні стилі керують організацією.

Програмне забезпечення повинно «легко розширювати свій функціонал, складатися з блоків і бути легким в обслуговуванні»,

Основними параметрами архітектури програмного забезпечення є якість роботи, низька відмовостійкість, масштабованість, надійність, реалізовуваність.

2.1.2 Клієнт-серверна архітектура

На сьогоднішній день, завдяки розвитку браузерів, а також засобів розробки продуктів на базі веб-технологій, з'явилася можливість створення повноцінних веб-додатків.

Веб-додаток – це програмне забезпечення, що працює на веб-сервері, на відміну від програмних засобів, що базуються на локальному збереженні в ОС пристрою. Доступ до веб-додатків здійснюється через веб-браузер із активним підключенням до Інтернету. Ці програми мають клієнт-серверну структуру [5] – користувачу (клієнт) надаються послуги окремим сервером. Приклади відомих веб-додатків включають в себе поштові клієнти (Gmail), сервіси роздрібних продаж в інтернеті (Eatsy, Amazon), інтернет-банкінг (Приват-24), тощо.

Проектування веб-сервісу передбачає розробку наступних складових:

- архітектури веб-додатку;
- протоколів зв'язку;
- сховищ даних.

Всі сучасні веб-додатки є розподіленими системами. Розподілена система – це набір незалежних комп'ютерів, які користу-вач сприймає як єдину об'єднану систему. Основними завданнями розподіленої системи є організація ефективного доступу користувачів до інформаційних і програмних ресурсів та ефективна взаємодія як користувачів з ресурсами, так і різних видів ресурсів між собою.

Відповідно, веб-додатки складаються з серверної та клієнтської частин. Клієнтська частина представляє користувачу інтерфейс взаємодії системи,

виконує первинну обробку даних, відповідає за логіку їх представлення. Серверна частина, в свою чергу, виступає у ролі оброблювача переданих через HTTP-запитів даних клієнтом та надає, за допомогою відповідних клієнтських запитів, сервіси у вигляді серверних операцій та відповіді на запити у вигляді фідповідного формату (JSON, HTML, XML, тощо). Відповідно можна зробити висновок, що клієнт-серверна модель передбачає функціонування кількох окремих додатків, які працюють на серверній та клієнтській частині відповідно. Діаграму розгортання розроблюваного додатку на базі клієнт-серверного підходу представлено на рисунку 1.

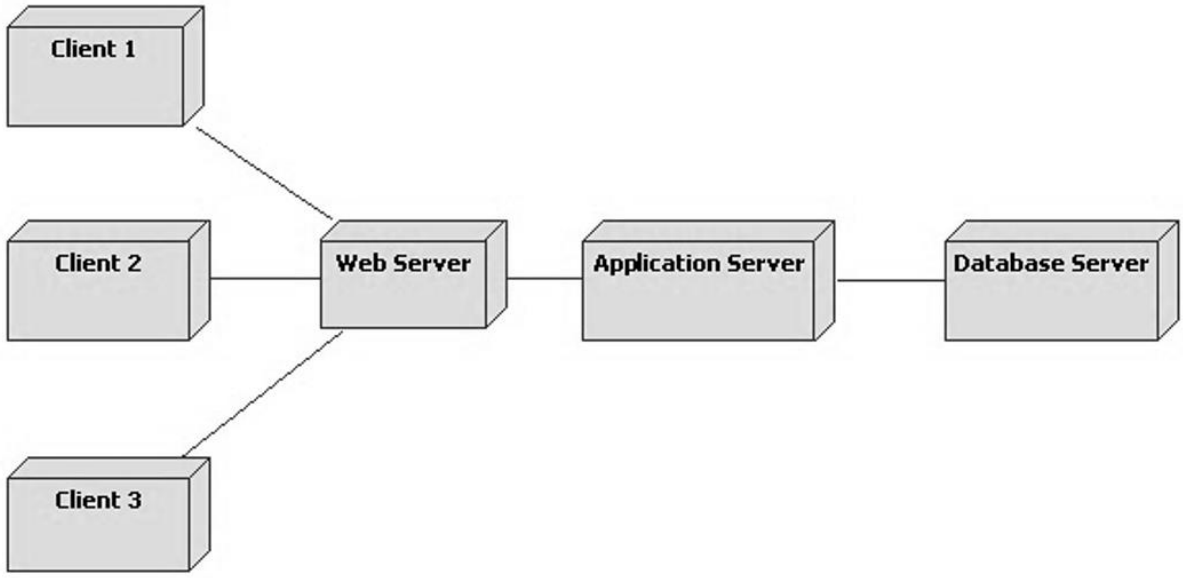


Рисунок 1. – Діаграма розгортання клієнт-серверної системи

Серверна частина призначена для обробки запитів великої кількості клієнтів одночасно. Клієнт в свою чергу завантажується в браузер кожного користувача і є набором одночасно функціонуючих екземплярів клієнтського додатку. Взаємодія між частинами системи відбувається за моделлю запит-відповідь через мережу. Відповідна структура системи поділяє зони відповідальності між незалежними спеціалізованими підсистемами і дозволяє модифікувати підсистеми окремо, без необхідності модифікації цілої системи, або припинення її функціонування.

Таким чином клієнт-серверна архітектура в рамках роботи веб-додатку передбачає використання в якості клієнта (ініціатора взаємодії) браузера, що завдяки протоколу прикладного рівня HTTP надсилає серверу (респонденту) запити (HTTP-requst) та отримує відповіді (HTTP-response). Даний зв'язок проїлюстровано на рисунок 2.

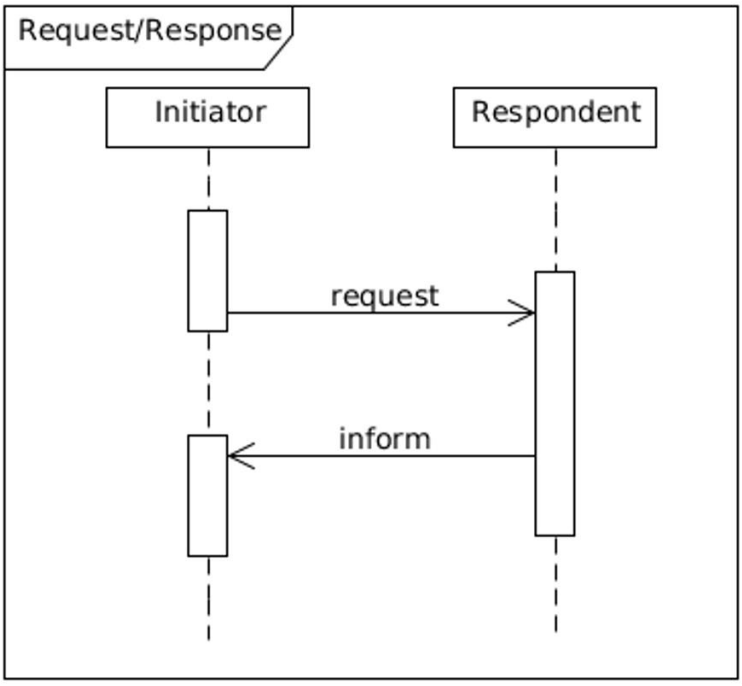


Рисунок 2. – Діаграма послідовності взаємодії клієнту (Initiator) та серверу (Respondent)

2.1.3 Класифікація веб-додатків

Веб-додатки можна класифікувати за наступними критеріями: [6]

- за використовуваним протоколом:
  - виключно HTTP. Кожен додаток використовує даний протокол для попереднього завантаження ресурсів клієнту;
  - FTP. Для передачі файлів, отримання безпосереднього доступу до ресурсів файлових систем, чи передачі даних у вигляді файлів;

- SSH Використовується для віддаленого підключення за допомогою встановлення захищеного з’єднання. Приклад – термінал віддаленого віртуального сервера в інтерфейсі користувача сервісу AWS;
- WebSockets. Використовується для взаємодії в реальному часі;
- комбіновані. Кожен додаток може використовувати комбінацію з перелічених протоколів;

- за типом клієнтського додатку:
  - веб-браузер;
  - окремий клієнт;
- за форматом серверної частини:
  - монолітний підхід;
  - мікросервісний підхід;
  - Serveless-підхід.

Проте особливе значення приділяється способу відображення клієнтських даних. Розрізняють два ключові підходи: серверного та клієнтського рендерингу сторінок. Розглянемо їх окремо на порівняємо зі звичайними сайтами.

2.1.3.1 Звичайні HTML-сайти

Хоча сьогодні багато сайтів створюються за допомогою компонентної основи, на базі фреймворків Angular, React або Vue, наряду з ними продовжують існувати звичайні сайти які базується на HTML-коді. Для таких сайтів, як правило, розробляється HTML-файл для кожного з маршрутів сайту. Коли користувач виконує запит на один із маршрутів, ваш повертає HTML-код для нього. Далі браузер аналізує цей код і надає вміст безпосередньо користувачеві. Загалом, процес розробки та використання виглядає приблизно так:

а) розробник будує HTML-сторінки, з використанням CSS, JS;

b) розробник розмішує файли сторінок на сервері;

c) клієнт завантажує HTML, CSS, JS файли з серверу;

d) клієнт одразу бачить вміст на екрані при запиті.

Відповідні компоненти зображені на схемі, представленій на рисунку 3.

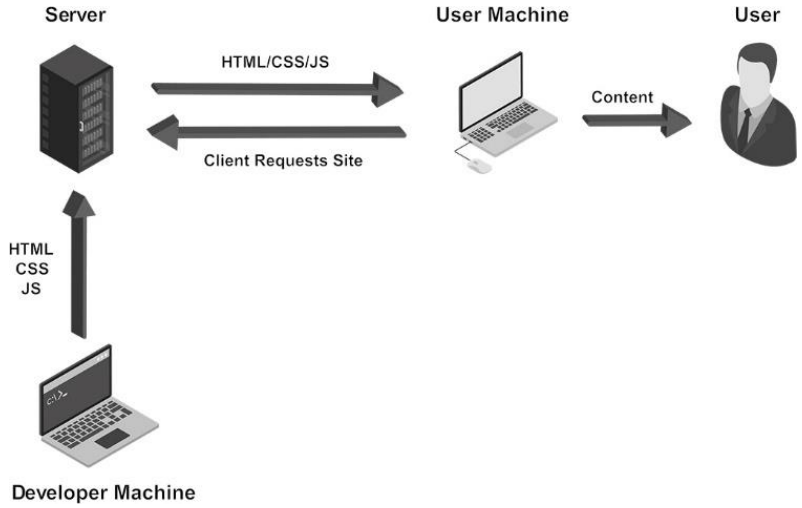


Рисунок 3. – Схема розгортання та використання системи на базу HTML-сайту

Це відносно простий процес і він доцільний для дуже невеликого об’єму ресурсів.

2.1.3.2 Рендеринг на боці клієнту CSR

Цей підхід використовується, за замовчуванням під час створення сайту на базі Angular, React або Vue. Давайте використаємо сайт на базі React як приклад. При побудові типового SPA React без використання таких фреймворків, як NextJS або Gatsby, життєвий цикл системи виглядає наступним чином:

a) розробник будує код React;

b) розробник завантадує скомпільовані файли на сервер;

c) клієнт завантажує код React з сервера;

d) код React запускається та генерує HTML / CSS на комп'ютері клієнта;
e) потім, після запуску React, користувачу відображається вміст додатку.

Відповідні елементи зображено на схемі, представленій на рисунку 4.

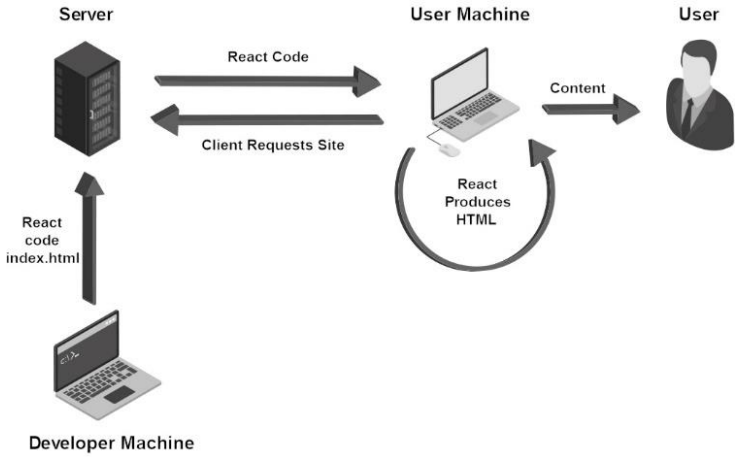


Рисунок 4. – Схема розгортання та використання системи на базі CSR

Код React повинен ініціалізуватися, щоб відобразити компоненти на екрані, перш ніж він зможе надати HTML-код браузеру для парсингу. Звичайно, є початковий файл HTML, який може містити індикатор завантаженн, але поки компоненти не ініціалізуються, вміст сторінки не буде корисним для користувача. Відповідний час завантаження може бути достатнім для середніх додатків, але при наявності великої кількості компонентів завантажуваних на екрані, можуть виникнути проблеми, з забезпеченням низького TTI і, відповідно користувацьким досвідом використання системи. Саме за таких сценаріїв часто використовується SSR.

2.1.3.3 Рендеринг на боці серверу SSR

Оскільки React потребує ініціалізаціїї, для кожного окремого входу на сторінку веб-сервісу необхідно завантажувати весь додаток. Аби вирішити проблему оцікування при першому запуску був розроблений підхід SSR. [7]

Для кожного запиту, який користувач відправляє на вервер за певним маршрутомвідбувається запуск екземпляру React. Далі формуються початкові візуалізації (так звані "fully hydrated") HTML та CSS, які вже готові для перегляду користувачем. Процес життєвого циклу системи виглядає змінюється на наступний:

a) розробник будує код React;
b) розробник завантажує скомпільовані файли на сервер;
c) клієнт виконує запит даних;
d) сервер запускає на сервері код React для створення HTML / CSS;
e) сервер надсилає згенерований HTML / CSS на екран;
f) користувач бачить вміст на екрані. React не потрібно запускати на комп'ютері користувача.

Дані компоненти зображені на схемі, що представлена на рисунку 5.

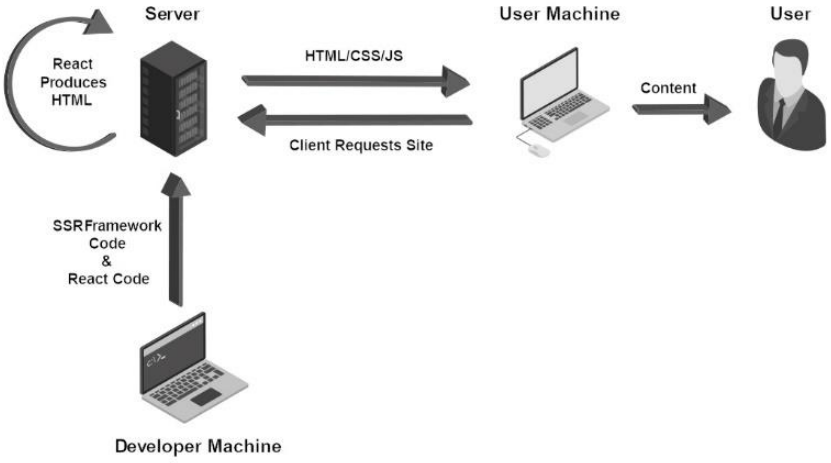


Рисунок 5. – Схема розгортання та використання системи на базі SSR

Окрім цього використання SSR дає кілька додаткових переваг. По-перше, мережеве з'єднання сервера, як правило, краще за з'єднання машини користувача, відповідно для виконання початкової реалізації запити ресурсів відбуваються швидше. Більше того, якщо сервер та база даних знаходяться у

межах одного хосту, можна повністю уникнути викликів поза локальних мереж для формування початкової візуалізації.

2.1.3.4 Генерація статичного сайту SSG

При використанні SSR, основне навантаження покладається на серверний додаток. Проте з використанням підходу SSG, навантаження на формування ресурсів відбувається одноразово на боці розробника, що значно здешевлює підтримку системи. Життєвий цикл системи включає наступні кроки:

a) розробник будує код React
b) розробник завантажує генерує HTML / CSS на своїй машині перед розгортанням на сервері
c) згенерований код розміщуєтсья на сервері
d) клієнт завантажує HTML, CSS, JS файли з серверу.
e) клієнт одразу бачить вміст на екрані при запиті

Відповідні компоненти зображені на схемі, представленій на рисунку 6.

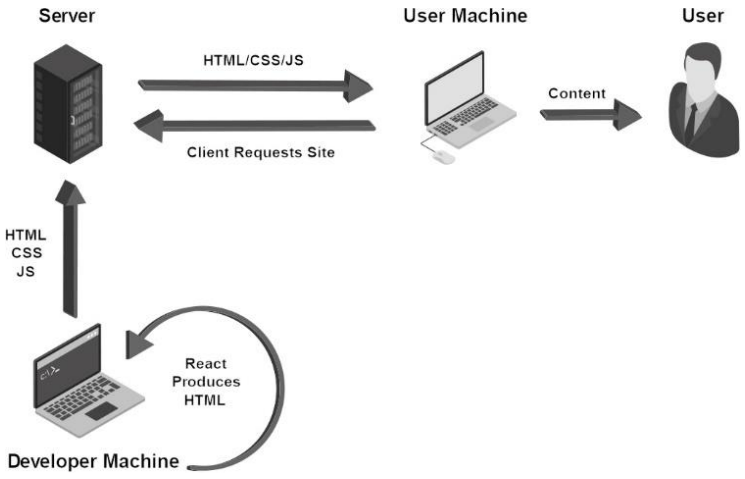


Рисунок 6. – Схема розгортання та використання системи на базі SSG

Даний підхід дозволяє розмащувати створені веб-клієнти як звичайні HTML-сторінки. Це значно знижує навантаження на сервер, збільшує швидкість надання сервером відповіді та повністю зберігає весь комплексний клієнтський функціонал представлений у додатку.

2.1.4 Дворівнева архітектура

Поняття дворівневої архітектури відображає підхід «thick client, thin server». [8] Будь-яка інформаційна система фактично повинна мати як мінімум три базові функціональні частини, а саме:

- модулі збереження даних;
- модулі обробки даних;
- інтерфейс користувача.

Відповідні компоненти повинні бути імплементовані незалежно. Система повинна давати можливість без зміни СУБД, і не зачіпаючи середовища серверної частини, зманювати користувацький інтерфейс

Відповідно, привки в користувацькому інтерфейсі, програмі представлення даних чи зміна платформи зберігання даних не повинні ніяким чином впивати на самі дані.

Проте двошарова архітектура окрім переваг має і недоліки. Мінімалізм архітектури забезпечує розробника і користувача необхідним функціоналом. Розподіл даних по окремим носіям і можливість одночасної обробки даних забезбечують недоторканість та цілісність даних. Однак використання даної архітектури має свої недоліки:

- необхідність використання проміжного вузла між клієнтом і сервером;
- недостатня потужність підтримки бізнес-логіки додатку і оновлення клієнту ускладнюють і перевантажують сервер.

2.1.5 Трирівнева архітектура

Поняття трирівневої архітектури відображає підхід «thin client – fat server» [9].

У відповідності до трирівневої архітектури, на клієті не представлений функціонал обробки даних. Клієт виконує лише власну функцію – представлення даних користувачеві, що надходять з основного серверу. Такий клієнт можна реалізувати за допомогую звичайних браузерних веб-технологій. Даний підхід зменшує обсях даних, що циркулюють між клієнтом та сервером, що в свою чергу дозволяє використовувати підхід у мережах з відносно повільним з’єднанням. Також клієнт може бути спрощений до вигляду звичайної веб-сторінки. Трирівнева архітектура клієнт-серверного додатку більш точно визначає ролі користувачів і дозволяє забезпечити безпеку доступу до даних і, відповідно, підвищує загальну захищеність системи від навмисного нападу, та помилок персоналу.

2.1.6 Архітектура REST

Аббревіатура означає Representational State Transfer - це стиль архітектури проектування вільно поєднаних додатків за допомогою протоколу HTTP, який часто використовується при розробці веб-сервісів. [10] REST не не передбачає правил впровадження нижнього рівня системи, проте представляє керівні принципи дизайну високого рівня системи і дозволяє обирати власну реалізацію.

REST визначає 6 архітектурних правил, які роблять будь-який веб-сервіс справжнім RESTful API:

- уніфікований інтерфейс;
- клієнт – серверна архітектура;
- відсутність стану;

- кешованість;
- багаторівнева архітектура;
- код за запитом (необов’язково).

2.1.6.1 Уніфікований інтерфейс

Як говорить сама назва правила, необхідно імплементувати інтерфейс доступу до ресурсів системи. Кожен ресурс в системі повинен мати один унікальний URI і повинен представляти можливість завантаження відповідної, або додаткової інформації. Загальний підхід передбачає синхронізацію ресурсу та веб-сторінки.

Будь-який єдиний ресурс не повинен бути занадто великим і містити увесь функціонал у своєму представленні. За необхідності, ресурс повинен містити посилання, що вказують на відносні URI, для отримання інформації, пов'язаної з цим.

Також представлення ресурсів у всій системі повинно дотримуватися конкретних вказівок, таких як конвенції іменування, формати посилань або формат даних (XML або / та JSON).

Усі ресурси повинні бути доступними через загальний підхід, такий як HTTP GET та аналогічно модифіковані за допомогою послідовного підходу.

Як тільки розробник ознайомиться з одним із API, він повинен мати можливість дотримуватися аналогічного підходу для всіх інших API.

2.1.6.2 Клієнт – серверна архітектура

Відповідне правило по суті означає, що клієнтська програма та серверний додаток обов’язково повинні мати можливість розвиватися окремо, не залежно одне від одного. Клієнт повинен знати лише URI- ресурсів, і нічого більше. Сьогодні це звичайна практика в розробці веб-сторінок.

Сервери та клієнти також можуть бути замінені та видозмінені окремо, доки інтерфейс між ними не буде змінений.

2.1.6.3 Відсутність стану

Всі взаємодії клієнту та серверу повинні бути виконанні без збереження стану. Сервер не повинен зберігати дані про останній запит HTTP, який зробив клієнт. Він повинен розглядати кожен запит як новий. Ні сесія, ні історія не допустимі.

Якщо клієнтській програмі потрібно стани для кінцевого користувача, де користувач входить у систему одночасно та після цього виконує інші санкціоновані операції, то кожен запит від клієнта повинен містити всю інформацію, необхідну для обслуговування запиту - включаючи аутентифікаційні та авторизаційні деталі.

Жоден контекст клієнта не повинен зберігатися на сервері між запитами. Відповідальність за управління станом програми лежить на клієнтській стороні.

2.1.6.4 Кешованість

У сучасному світі кешування даних та серверних відповідей є надзвичайно важливим, де б вони не застосовувалися. Кешування поліпшує продуктивність клієнта та покращує масштабованість для сервера, оскільки з провадженням кешування, навантаження на сервер зменшується.

У підході REST кешування застосовується до ресурсів, коли це можливо, і тоді ці ресурси обов’язково оголошуються кешованими. Кешування може бути реалізовано на стороні сервера або клієнта.

Добре налаштоване кешування частково або повністю виключає деякі клієнт-серверні взаємодії, ще більше покращуючи масштабованість та продуктивність.

2.1.6.5 Багаторівнева архітектура

REST дозволяє використовувати багаторівневу системну архітектуру, де, наприклад API розгортуний на сервері А, дані зберігаються на сервері В і аутентифікаційні запити, обробляються, наприклад, на сервері С. Клієнт не може визначити, підключений він безпосередньо до кінцевого сервера чи до посередника по шляху, оскільки серверна частина представлена розподіленою системою.

2.1.6.6 Код за запитом

Дане правило є опціональним. Більшість запитів несуть навантаження у вигляді статияного представленя даних форматами XML або JSON. Але за необхідності, підхід REST передбачає повернення виконуваного коду для підтримки частини функціоналу програми. Наприклад, клаєнт може виконати запит до серверу для отримання програмного коду відтворення користувацького інтерфейсу.

2.2 Вибір фреймворків для розробки клієнтської частини

2.2.1 Агрументування використання фреймворків пи побудові клієнтської частини систем

Frontend - це частина веб-сайту, яку бачать користувачі у своєму браузері. Відповідно, backend – серверна сторона системи і, як правило,

відповідає за бізнес-логіку веб-додатку. [11] Frontend складається з трьох невід'ємних складових:

- HTML – мова розмітки, що визначає структуру веб-сторінки;
- CSS – таблиці стилів, котрі визначають зовнішній вигляд структурних елементів;
- JavaScript - мова програмування, яка описує функціональність та відповідає за динамічні елементи на веб-сторінці.

Технічно всі три елементи - це лише рядки коду, які потім обробляються браузером для відображення веб-сторінки. Розробники витрачають багато часу та ресурсів на написання коду, для забезпечення мінімального функціоналу та прийнятного зовнішнього вигляду. Однак фреймворки можуть зменшити кількість необхідного коду та оптимізувати процес розробки в цілому. Це призводить до скорочення часу розробки та витрат на виведення товару на ринок.

Програмний фреймворк (frontend або backend) включає стандартизований попередньо написаний код, що полегшує та швидше розвиває певну функціональність.

Вибір фреймворку – непросте питання. Є багато характеристик та факторів, які слід врахувати, з різними перевагами та недоліками для порівняння. Тому було підготовано детальне порівняння найкращих фреймворків мови JavaScript.

Аргументацією до вибору мови стало те, що це одна з найпопулярніших мов програмування у світі та найпоширеніша технологія побудови веб-інтерфейсів.

Як правило, код frontend-частини можна записати будь-якою мовою програмування. Однак цей код повинен бути перетворений у JavaScript, оскільки браузери наразі можуть виконувати лише JS. Впровадження WebAssembly, одного з основних напрямків розвитку веб-сайтів, може

незабаром змінити цю ситуацію. Однак JavaScript все ще є оптимальним вибором для розробки інтерфейсів.

2.2.2 Аналіз та порівняння сучасних фреймворків для розробки веб-додатків

При розробці клієнтської частини веб-орієнтованого додатку постає питання вибору технології, яка дозволить імплементувати клієнтську частину багаторівневої клієнт-серверної архітектури на базі RESTful API та CSR-підходу SPA.

Існує багато фреймворків JavaScript, які полегшують та прискорюють розробку на стороні клієнта. Для порівняння було обрано три найпопулярніші та найефективніш: Angular, React та Vue.js.

2.2.2.1 Аналіз фреймворку Angular

Фреймворк AngularJS був вперше випущений компанією Google у 2010 році. У 2016 році з'явився Angular 2, який був повною переробкою Angular JS. Відтоді фреймворк часто оновлється. Angular 9 - це остання доступна версія.

Переваги фреймворку Angular:

- структура MVC. Angular використовує підхід MVW (Model-View-Whatever), яка традиційно використовується як MVC (Model-View-Controller). Завдяки цьому додаток ділиться на три взаємопов’язані компоненти. Це дає можливість розробникам писати добре структурований код, що є значною перевагою для розробки складних проектів;
- шаблони Angular. Шаблони для створення компонентів є дуже читабельнми відноно інших фреймворків, оскільки в основному вони використовують стандартні теги HTML;

- проста реалізація двосторонньої прив'язки даних. Двостороння прив'язка даних означає, що будь-які зміни моделі впливають на представлення. І навпаки, коли представлення змінюється, модель відразу ж змінюється. Angular дозволяє використовувати просте двостороннє прив'язування даних, що вигідно для простих додатків. Більш складні додатки швидше працюють з односторонньою прив'язкою даних, яка працює лише в одному напрямку (view-to-model або model-to-view), залежно від потреб програмного забезпечення. Це дозволяє зберегти ресурси;

- велике співтовариство. Angular має велике співтовариство, яке розвивалося з моменту випуску AngularJS і стало сильнішим з моменту виходу Angular 2. Фреймворк має близько 500 000 завантажень щотижня на npm та понад 45 тисяч зірок на Github. Ця популярність означає, що існує ряд рішень, сумісних з різними версіями Angular, а також можливість отримувати поради досвідчених розробників та користувачів, не потребуючи звернення до офіційної служби підтримки.

Недоліки фреймворку Angular:

- не використовується Shadow DOM як типове. Shadow DOM вирішує проблему унікальних імен елементів сторінки чи ідентифікаторів, що може створити розробникам серйозні проблеми, особливо якщо мова йде про складні проекти. Розробник може змінити стиль CSS одного файлу, а інші файли також будуть під впливом тих самих змін. Shadow DOM дозволяє веб-браузерам включати під-дерево елементів DOM у візуалізацію документів, але не в основний документ DOM. Shadow DOM інкапсулює стилі, сценарії та вміст всередині спеціального елемента, щоб вони не заважали іншому вмісту в додатку. У випадку з Angular 2+, тіньовий DOM увімкнено у веб-браузерах, які підтримують його. Інакше інкапсуляція Shadow DOM імітується;

- не використовується віртуальний DOM. Віртуальний DOM - це спрощена копія DOM. Віртуальний DOM дозволяє швидко змінювати будь-який елемент без необхідності рендерингу всього DOM. Цей підхід є

тенденцією серед сучасних фреймворків веб-розробки, але, на жаль, його ще не підтримує жодна версія Angular. Натомість Angular 2 використовує односпрямований потік даних для виявлення змін у моделі та оновлює лише ті частини DOM, на які впливає зміна моделі;

- використання TypeScript. Будучи фреймворком JS, Angular підтримує використання чистого JavaScript. Однак цей фреймворк був створений для використання з TypeScript, надбудовою над JavaScript, створеною Microsoft. TypeScript дозволяє розкрити додаткові можливості Angular;
- низька швидкість візуалізації. Обмежене використання тіньового DOM та відсутність віртуальних DOM призводить до зниження продуктивності. Швидкість візуалізації / повторного відображення перегляду повільніша порівняно з продуктивністю інших фреймворків JavaScript;
- великий об’єм коду. Angular - це комплексна структура із великим обсягом коду, який потрібно завантажити з сервера, перш ніж побачити веб-додаток у своєму браузері. Як результат, швидкість та продуктивність знижуються. На щастя, ці проблеми можна вирішити за допомогою tree-shaking - техніки, яка дозволяє усунути невикористаний код у додатку. Webpack - ідеальне середовище для tree-shaking. Ось як це працює: Webpack сканує код, розміщує всі модулі в одному файлі та видаляє весь "експорт" з коду, який не імпортується. Потім потрібно запустити процес мінімізації. Як результат, пакет позбавляється від невикористаного коду.

Angular продовжує розвиватися, незважаючи на те, що фреймворку вже більше 10 років.

Найкращі варіанти використання Angular:

- розробка веб-платформ. Angular 2+ був створений з урахуванням використання мобільних пристроїв. Усі обмеження, які мають мобільні пристрої (такі як навігація через сенсорний екран, невеликий розмір екрана та мобільне обладнання), були враховані в нових версіях фреймворку. Ось чому, думаючи про підхід mobile-first, обирають Angular;

- програмне забезпечення підприємства. Angular відмінно підходить для додатків зі складною архітектурою, наприклад, корпоративного програмного забезпечення;
- розробка прогресивних веб-додатків та гібридних мобільних додатків. Angular 9 та Ionic 4 в даний час ідеально підходять для створення гібридних додатків, які працюють на різних мобільних платформах. Цей же стек технологій можна використовувати для прогресивної розробки веб-додатків. Angular, як правило, дуже зручний для мобільних пристроїв завдяки Angular Mobile Toolkit, який дозволяє швидко та просто розробити PWA, гібридні програми та мобільні веб-сайти.

Кілька додатків, відомих у всьому світі, були побудовані за допомогою Angular, і для цього є вагомі причини. Серед них:

- PayPal. PayPal - один з найпопулярніших сучасних платіжних засобів. Він забезпечує інтуїтивно зрозумілі і прості розрахунки, які були розроблені за допомогою Angular;
- The Guardian. Британська щоденна газета The Guardian впровадила кращі практики UI на своєму веб-сайті за допомогою Angular. Цей фреймворк разом із розширеннями RxJS, наприклад, відповідає за нескінченне прокручування сторінки;
- Strapping - система управління замовленнями. Strapping - це власне програмне забезпечення, яке використовується для складного управління запасами. Angular тут був використаний як частина стека MEAN разом з MongoDB, Express.js та Node.js;
- Toddy - гібридний мобільний додаток. Toddy - це інтернет-ринок для батьків та нянь. Щоб скоротити час і витрати на розробку, був розроблений крос-платформне додаток, використовуючий комбінацію Ionic 2 та Angular 2.

2.2.2.2 Аналіз фреймворку React

React - це фактично бібліотека JavaScript, створена для побудови інтерфейсів користувача. Він підтримується Facebook та Instagram і став основною технологією для нескінченного скролу в цих двох додатках. Як бібліотека JS, React має обмежену сферу використання, але в комплекті з іншими бібліотеками він стає потужним рішенням, що робить його головним конкурентом Angular.

Переваги React:

- компонентна модель. React не використовує жодних шаблонів. Логіка компоненту написана на JavaScript, що забезпечує їй більшу гнучкість та дозволяє великим обсягам даних легко проходити через додаток, зберігаючи стан DOM. Незважаючи на те, що цей підхід використовується у всіх порівнюваних frontend фреймворках, React першим запровадив компонентну модель;
- віртуальний DOM. Як було пояснено у огляді Angular, віртуальний DOM дозволяє створити спрощену копію DOM. Усі зміни, які потрібно здійснити, вносяться у віртуальний DOM. Далі обидва DOM порівнюються, і, при виявленні відмінностей, реальний DOM буде відображати лише змінену частину. Цей процес набагато швидший та ефективніший, ніж робота безпосередньо з DOM;
- одностороння прив'язка даних. Двостороння прив'язка даних була перевагою для Angular, і одностороння прив'язка даних React також може бути перевагою. Такий підхід змушує представлення реагувати на будь-які зміни, внесені в модель, але зміни в самому представленні не можуть вплинути на модель. В результаті дані протікають лише в одному напрямку, зменшуючи можливість виникнення будь-яких побічних ефектів;
- використання чистих функцій. На відміну від Angular, React не вимагає використання класів. Інтерфейс програми можна створити за допомогою чистих функцій, спрощуючи базу даних коду;

- нативне середовище розвробки мобільних додатків. React Native - це часткова надбудова React, створена для розробки власних мобільних додатків за допомогою JavaScript. З його допомогою програми можуть включати в себе вбудовані блоки і виглядати точно так само, як додатки, побудовані за допомогою Objective-C або Java. Велика перевага полягає в тому, що потрібен лише один розробник, щоб створити нативну програму для двох платформ, оскільки для цього потрібні лише знання JavaScript. У порівнянні з гібридними мобільними додатками, побудованими за допомогою Angular, додатки React Native мають значно більшу продуктивність і майже такі ж потужні, як і рідні програми;

- велике співтовариство. За підтримки двох найбільших світових компаній у галузі соціальних медіа, React має мільйони користувачів у всьому світі. Ця сильна спільнота означає, що користувачі мають доступ до швидкої допомоги досвідчених розробників. На Github React зібрано понад 122 тисячі зірок, а статистика використання npm показує, що фреймворк завантажується більше 5 мільйонів разів на тиждень.

Недоліки React:

- необхідність використання сторонніх технологій. Як вже було сказано, React - це бібліотека JavaScript, а не фреймворк. В результаті його можливості обмежені. Для розширення функціональності необхідно використовувати сторонні модулі та бібліотеки. Читання документації та навчання використанню цих технологій може зайняти деякий час;

- використання JSX. React рекомендує використовувати JSX замість звичайних JavaScript та HTML. Це насправді JavaScript, але він розширений синтаксисом XML. JSX швидше, безпечніше і простіше, ніж JS. Отже, щоб отримати максимальну користь від React, необхідно вивчити цю модифікацію JavaScript;

- складна структура додатків. React не має заздалегідь визначеної структури. Це означає, що за структурування програми повністю покладається відповідальність розробника, роблячи це залежним від їх знань та досвіду;

- крута крива навчання. React - це не найпростіша технологія вивчення. Крім самого React, для побудови додатку також необхідно ознайомитися з кількома іншими бібліотеками та модулями.

React досить універсальний, це означає, що з використанням даної технології можна побудувати майже будь-який клієнтський додаток. Однак є деякі випадки, коли React є незамінним. Для наступних додатків React є найкращим рішенням, доступним на ринку:

- динамічні додатки. React є хорошим вибором для створення додатків, що потребують швидкого відтворення, оскільки дозволяє швидко відображати зміни даних у представленні;

- односторінкові програми. Оскільки React може відображати зміни вмісту, не перезавантажуючи поточну сторінку, це прекрасне рішення для односторінкових програм;

- нативні мобільні додатки. React Native дозволяє створювати мобільні додатки, які не відрізняються від нативних програм, створених за допомогою Java або Objective-C.

React був розроблений та підтриманий Facebook та Instagram, і ці два додатки для соціальних медіа є класичним прикладом використання React:

- Facebook. Не дивно, що Facebook використовує власну технологію у своїх продуктах. Наприклад, React Native - це ключова технологія, яка використовується в додатку Ads Manager – мобільному веб-сайту і додатку, що дозволяє клієнтам керувати своїми оголошеннями в Facebook на ходу. Це перший кросплатформенний додаток React Native;

- Instagram. Веб-версія Instagram також була повністю побудована з використанням React. Бібліотека JavaScript забезпечує оновлення вмісту, не

потребуючи перезавантаження сторінки, саме так працює Instagram-канал. Мобільний додаток Instagram використовує React Native.

2.2.2.3 Аналіз фреймворку Vue

Vue - популярна альтернатива Angular and React. Цей прогресивний фреймворк для побудови інтерфейсу набирає популярності. Спочатку він став надзвичайно популярним у Китаї, а зараз стає популярним і на Заході.

Переваги Vue:

- структура MVC. Так само, як Angular, Vue є фреймворком, що використовує підхід MVC (або Model-View-Controller). Перевага цього очевидна - це дозволяє написати добре структурований код, що вкрай важливо при розробці складних додатків;
- рішення з малим обсягом коду. Важливою перевагою Vue є невеликий розмір, оскільки він не включає багато функцій прямо «з коробки», але функціональність легко розширюється різноманітними сторонніми рішеннями. Його часто порівнюють з Angular, який представляє собою монолітний фреймворк, який має купу вбудованих функцій, які навряд чи взагалі будуть використані у додатку. Звичайно, tree-shaking дозволяє усунути невикористаний код, але фреймворку все ж більший порівняно з тим, що пропонує Vue;
- декларативні шаблони. Шаблони на Vue.js написані мовою HTML, що робить їх читабельними без знання інших мов розмітки, або програмування;
- віртуальний DOM. Завдяки більш легкій віртуальній реалізації DOM, додатки, створені за допомогою Vue.js, мають найвищу ефективність порівняно з іншими frontend фреймворками;
- двостороння прив'язка даних. Vue.js автоматично синхронізує всю модель даних із DOM;

- чистий JavaScript. Vue.js використовує чистий JavaScript, позбавляючи від необхідності розробників або тестових інженерів вивчати будь-яку іншу мову програмування;

- зростаюча популярність. Маючи близько 800 000 завантажень за тиждень, Vue.js має чудові результати. Понад 128 тис. Зірок Github демонструють зростаючу популярність і відображають погляди співтовариства. Vue.js зараз використовується у всьому світі. Це означає, що рішення, сумісні з цією системою JS, стануть все більш доступними;

- полога крива навчання. Порівняно з Angular та React, Vue.js - це найпростіша технологія розробки веб-інтерфейсів, яку можна вивчити. Крім того, впровадження Vue.js до проекту може бути поетапним.

Недоліки Vue:

- мале співтовариство. Vue є менш популярним у порівнянні з Angular та React, обидва вони вражають кількістю користувачів;

- менша кількість бібліотек для Vue.js. Оскільки користувачів менше, для розширення функціональності фреймворку було розроблено менше рішень;

- китайське походження. Документація для деяких сторонніх бібліотек може бути доступна лише китайською мовою. Але офіційна документація повністю доступна англійською мовою.

Vue може використовуватися для побудови багатьох типів додатків. Завдяки сумісності з іншими бібліотеками JavaScript та здатності додавати складніші логіки до існуючих додатків, він може забезпечити ідеальне рішення майже для будь-якого типу проекту. Однак Vue.js має розглядатися як основний варіант, для розробки наступних типів додатків:

- динамічні високопродуктивні додатки. Подібно до React, Vue.js - хороший вибір для динамічних додатків, однак завдяки віртуальному DOM він пропонує більш високу продуктивність, що вигідно для складних додатків;

- односторінкові додатки. Vue.js дозволяє швидко оновлювати вміст, не завантажуючи сторінку, що робить цей фреймворк ідеальним для SPA.

Хоча Vue є відносно новою технологією, вона вже використовується в кількох масштабних проектах, переважно в Китаї.

Веб-сайти для Alibaba, Xiaomi, Baidu і Tencent використовують Vue як центральну технологію інтерфейсу.

2.2.3 Аргументація вибору стеку технологій клієнтської частини

Під час розробки технічного рішення для клієнтської частини було розглянуто найпопулярніші фареймворки для розробки односторінкових веб додатків, а саме Vue.js, Angular та React.

Щоб зробити огляд frontend фреймворків легшим для розуміння, в додатку Б приведено таблицю, яка порівнює критичні фактори, які слід враховувати при виборі між ними.

У результаті порівняння були зроблені висновки, що для розробки об’ємного проекту у якості основи найкраще підходить фреймворк Angular. Він забезпечує максимальну гнучкість та швидкість відмалювання сторінки. Абсолютна більшасть недоліків фреймворку було вирішено з виходом та вдосконаленням версій після Angulae 2. Великий досвід інших розробників дозволяє вирішити проблеми, які обов’язково виникають при розробці комплексного додатку, незалежно від технологій. React у цьому випадку виявляється надто об’ємним і для вирішення кожної окремої задачі потребує окремої бібліотеки, а для Vue.js ще не існує великої калькості гайд-лайнів.

Для опису стилів у представленнях вирішено використати препроцесор SCSS, який вирішує наступні труднощі великих проектів:

- Трудомісткість внесення невеликих змін;
- Труднощі структурування коду;
- Надмірність коду;

- Велика довжина строк CSS-класів та правил.

Як мову розмітки шаблонів, обрано HTML. Для синхронізації проекту та контролю версій обрано розподілену систему керування версіями файлів та спільної роботи Git. Репозиторій розміщено на платформі Gitlab.

2.3 Вибір технології збереження даних

Протягом вирішення завдання по збереженню даних тестів та користувачів постає питання вибору між SQL та NoSQL базами даних. [12]

2.3.1 Структура даних

Першим і головним фактором прийняття рішення щодо вибору між SQL та NoSQL базами даних є те, якку структуру мають дані.

Якщо дані є структурованими, то, кращим вибором є SQL-база даних. SQL Бази даних чудово підходить для орієнтованих на трансакції систем, таких як інструменти управління відносинами з клієнтами, програмне забезпечення бухгалтерського обліку та платформи електронної комерції. Кожен рядок у базі даних SQL є окремим об'єктом (наприклад, клієнтом), і кожен стовпець є атрибутом, який описує цю сутність (наприклад, адреса, назва роботи, придбаний предмет тощо).

Через такі чіткі структуровані зв'язки між рядками та стовпцями таблиці, бази даних SQL підходять краще, коли постає необхідність відповідності принципам ACID. [13]

ACID розшифровується як:

- атомарність (Atomicity) - кожна транзакція або повністю виконується, або повністю відкочується назад;
- консистентність (Consistency) - дані, записані в базу даних, повинні задовольняти всі визначені правила;

- ізоляція (Isolation) - коли транзакції проводяться одночасно, вони не конкурують між собою, а діють так, ніби виконуються послідовно;
- довговічність (Durability) - Після того, як транзакція була здійснена в базі даних, вона вважається постійною, навіть у разі відмови системи.

Відповідність ACID захищає цілісність даних, визначаючи, транзакцію та те як вона взаємодіє з базою даних. Це дозволяє уникнути несинхронізації таблиць баз даних, що дуже важливо, наприклад, для фінансових операцій. Відповідність ACID гарантує дійсність транзакцій навіть за умови помилок, технологічних збоїв, катастрофічних подій тощо.

Якщо дані дуже структуровані і відповідність принципам ACID необхідна, SQL - чудовий вибір.

З іншого боку, якщо вимоги до даних не чіткі або якщо дані неструктуровані, NoSQL може бути найкращим варіантом.

Дані, які зберігаються в базі даних NoSQL, не потребують заздалегідь визначеної схеми, як для бази даних SQL. Вірніше, дані можуть бути сховищами стовпців, орієнтованими на документи, на основі графіків або парами «ключ-значення». Це забезпечує набагато більшу гнучкість та зменшує об'єм ахітектурного планування під час керування базою даних.

На рисунку 7 зображені основні види структур баз даних, які відповідно поділені по характеристиці приналежності типам SQL та NoSQL.

З використанням NoSQL бази даних відкриваютсья наступні можливості:

- створення документів, не ретельно визначаючи їх структуру наперед;
- додавання полів до бази даних, не змінюючи полів існуючих документів;
- збереження документі, які мають свою унікальну структуру;
- утримання кількох баз даних з різною структурою та синтаксисом.

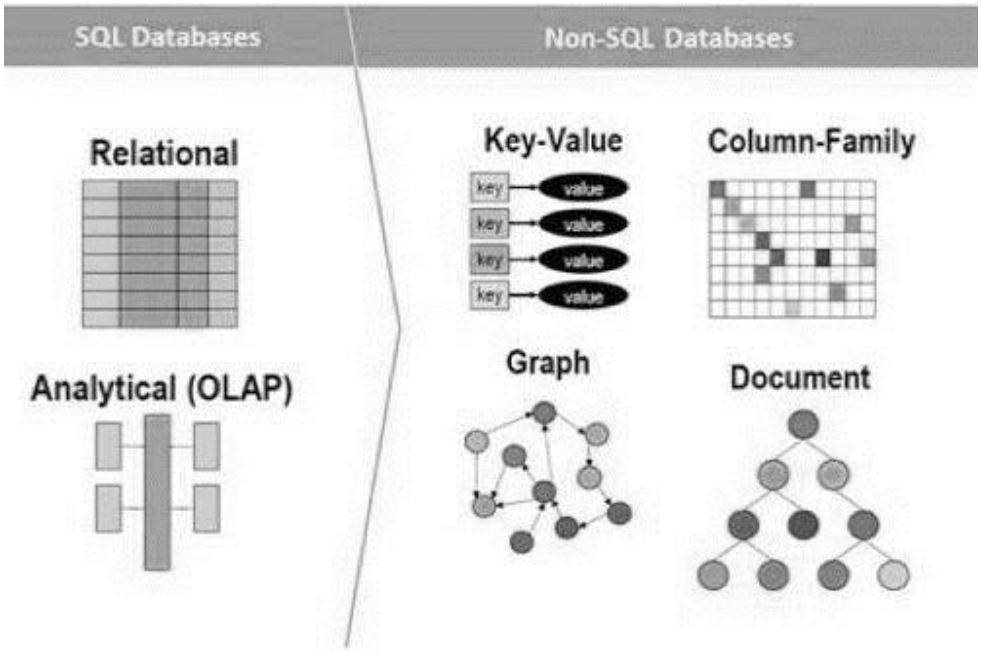


Рисунок 7. – Бази даних SQL та NoSQL

Бази даних NoSQL набагато краще підходять для зберігання таких даних, як вміст статті, публікації в соціальних мережах, даних датчиків та інших типів неструктурованих даних, які не вписуються в рамнки табличного підходу. Бази даних NoSQL були створені з урахуванням гнучкості та масштабованості, яка відповідає моделі BASE-узгодженості[4], що означає:

- базова доступність (Basic Availability). Це означає, що хоча база даних гарантує доступність даних, база даних може не отримати запитувані дані, або дані можуть перебувати в стані, що змінюється, або в непослідовному;
- м'який стан (Soft state). Стан бази даних може змінюватися з часом;
- побічна послідовність (Eventual consistency). База даних з часом стане послідовною, і дані будуть розповсюджуватися скрізь у якийсь момент майбутнього.

Модель BASE була побудована для максимальної гнучкості. Але є деякі бази даних NoSQL, сумісні з ACID.

Структура даних є найважливішим фактором у вирішенні питання про те, чи використовувати SQL чи NoSQL базу даних.

2.3.2 Можливість запиту даних

Наступним фактором, який слід врахувати, є те, як часто будуть відбуватися запити даних, як швидко повинні запускатися запити та що буде відповідати за виконання цих запитів.

Оскільки дані добре структуровані та організовані, дуже ефективно запитувати їх за допомогою бази даних SQL.

SQL - популярна мова програмування, яка існує вже більше 45 років, тому вона надзвичайно зріла і добре відома. Мова ефективно виконує запити та швидко витягує та редагує дані. Вона має дуже легкий і декларативний характер, і тому її легко вивчити. Тому запити можуть виконувати за використанням меншого технічного персонал, як бізнес-аналітики та маркетологи.

База даних NoSQL забезпечує гнучкість у використанні різних типів даних які зберігаються, але через потенційно великі відмінності в структурах даних, запит не настільки ефективний, як у SQL базах даних.

Коли розроблялася технологія баз даних NoSQL, розробники зосереджувались на масштабованості та гнучкості, а не на ефективності запитів.

Тому, для підготовки запуску запитів NoSQL доведеться виконати додаткову обробку даних. Залежно від бази даних NoSQL, яка використовується, зазвичай, доводиться реалізовувати певний рівень MapReduce. [5] Більшість розробників будують функціональні запити в шарі додатків, та не турбуються про це в шарі бази даних. Були спроби стандартизації запитів NoSQL, наприклад, XQuery або JSONiq, але ці інструменти не отримали широкого поширення.

Запити бази даних NoSQL зазвичай вимагають розробників або науковців, спеціалізованих на роботі з даними, що значно дорожче і менш ефективно.

2.3.3 Масштабування

Бази даних SQL і NoSQL змінюються по-різному, тому доведиться подумувати те, як зростатиме набір даних у майбутньому.

Бази даних SQL масштабуються вертикально, тобто потрібно збільшити ємність одного сервера (збільшуючи потужність процесора, об'єм оперативної пам’яті або SSD) для масштабування бази даних. Бази даних SQL були розроблені для роботи на одному сервері для підтримки цілісності даних, тому їх непросто масштабувати.

Бази даних NoSQL масштабуються горизонтально, це означає, що присутня можливість додання більшої кількості серверів для живлення зростаючої бази даних. Це величезна перевага, яку NoSQL має над SQL.

Здатність баз даних NoSQL горизонтально масштабуватися пов'язана з відсутністю структури даних. Оскільки NoSQL вимагає значно меншої визначеності структури, ніж SQL, кожен збережений об'єкт є майже самостійним та незалежним. Таким чином, об'єкти можуть бути легко збережені на декількох серверах без необхідності з'єднання. Це не стосується SQL, де кожен рядок та стовпець таблиці мають бути пов'язані.

2.3.4 Конвергенція SQL і NoSQL

І бази даних SQL, і NoSQL мають свої плюси і мінуси. Таким чином, з'явилася можливість виокремлення найкращих характеристик обох типів баз даних та інтегрування їх.

Наприклад, найпопулярніша реляційна база даних з відкритим кодом MySQL пропонує MySQL Document Store. Це забезпечує структуру бази даних MySQL у поєднанні з гнучкістю та високою доступністю NoSQL без необхідності впровадження окремої бази даних NoSQL.

MongoDB, одна з найпопулярніших баз даних NoSQL, пропонує багатодокументні транзакції ACID.

База NoSQL, керована AWS, DynamoDB, також забезпечує функціональність транзакцій, сумісних з ACID.

Завдяки простому налаштуванню баз даних, які пропонують постачальники хмарних послуг, з'являється можливість використовувати як бази даних SQL, так і NoSQL при використанні архітектури хмарних даних.

2.3.5 Огляд баз даних

До локальних варінтів баз даних SQL належать:

- MySQL - як згадувалося раніше, найпопулярніша реляційна база даних з відкритим кодом;
- Microsoft SQL сервер - корпоративна версія Microsoft SQL;
- PostgreSQL - база даних з відкритим кодом на рівні підприємства, орієнтована на розширюваність;
- Oracle - повноцінний (і дорогий) варіант SQL;
- MariaDB - розширена версія MySQL, побудована оригінальними розробниками MySQL;
- Та багато інших.

Основні платформи хмарних сервісів які мають власні варіанти SQL:

- AWS має:
  - RDS, стандартна хмарна база даних SQL;
  - Aurora, яка зосереджена на підвищеній пропускній здатності та масштабованості;
- Microsoft Azure має:
  - База даних SQL Azure, власне керована база даних як послуга;
  - База даних Azure для MySQL, PostgreSQL та MariaDB;
- Google Cloud Platform (GCP) має:

- Хмарний SQL, який можна використовувати для MySQL та PostgreSQL;
- Cloud Spanner, який поєднує елементи SQL і NoSQL.

До локальних варіантів баз даних NoSQL належать:

- MongoDB - найпопулярніша база даних NoSQL;
- Redis - відкрита, розповсюджена, в пам'яті база даних ключових значень, яка є надшвидкою;
- Cassandra - безкоштовна база даних NoSQL з відкритим кодом, створена Facebook, яка фокусується на масштабованості та високій доступності;
- багато інших.

Також постачальники хмарних послуг пропонують безліч варіантів NoSQL:

- AWS має:
  - DynamoDB, її керована база даних NoSQL;
  - DocumentDB - швидка, масштабована, високодоступна база даних MongoDB;
- Microsoft Azure пропонує mosDB, її глобально розповсюджена багатомодельна база даних;
- Google Cloud має:
  - Bigtable, послуга бази даних із широкими колонками NoSQL;
  - Cloud Datastore, його служба бази даних NoSQL;
  - Cloud Firestore - хмарна база даних документів NoSQL, яка допомагає зберігати та запитувати дані додатків.

2.3.6 Аргументація вибору технології збереження даних

Бази даних SQL надають великі переваги транзакційним даним, структура яких не змінюється часто (або зовсім) і де цілісність даних є

найважливішою. Вони також найкраще підходять для швидких аналітичних запитів.

Бази даних NoSQL забезпечують набагато більшу гнучкість та масштабованість, піддаються швидкому розвитку та ітерації.

З урахуванням використання строго типізованої мови програмування для реалізації серверної частини та наявності складних взаємо пов’язаних сутностей, було зроблено вибір SQL бази даних.

Відпоідно до розглянутих СУРБД та їх характеристик, у результаті було обрано Microsoft SQL Server оскільки дана СУБД має ряд переваг відносно інших SQL СУБД, а саме:

- продукт простий у використанні;
- поточна версія працює швидко та стабільно для будь-яких задач;
- основа дає можливість регулювати і відслідковувати рівні продуктивності, які допомагають знизити використання ресурсів;
- дана СУБД добре взаємодіє зі стеком Microsoft, який використовується для побудови серверної частини.

3 ВЛАСНА РЕАЛІЗАЦІЯ

3.1 Проектування бази даних

В рамках реалізації функціоналу, передбаченого у розділі аналізу предметної області при формуванні деталізації технічного завдання та технічних можливостях рішення по технології збереження даних навеедних у розділі пошуку технічного рішення, були виділенні наступні блоки даних, які потребують зберігання:

- дані користувача;
- дані про роль користувача;
- дані необхідні для роботи системи автентифікації та авторизації;
- дані тестів;
- дані про історію проходження тестів;
- дані для класифікації тестів;
- питання тестів;
- відповіді на тест;
- службові дані роботи систем.

У відповідності до перелічених блоків та логічних зв’язків між ними було розроблено базу даних, архітектура якої зображена на діаграмі, представленій на рисунку 8.

Далі розглянемо кожну таблицю даної бази даних отремо.

Таблиця AspNetUsers представляє сутність користувача системи. Ключем є поле ідентифікатора. Поле UserName представляє ім’я користувача.

Поле NormalizedUserName представляє нормалізовану форму ім’я. Використовується для прискорення виконання запитів вибірки за ім’ям (наприклад для перевірки наявності користувача в системі, знаходження колізій імен) та викликає зменшення навантаження на сервер у майбутньому за рахунок відсутності необхідності кормалізувати ім’я при кожній вибірці.

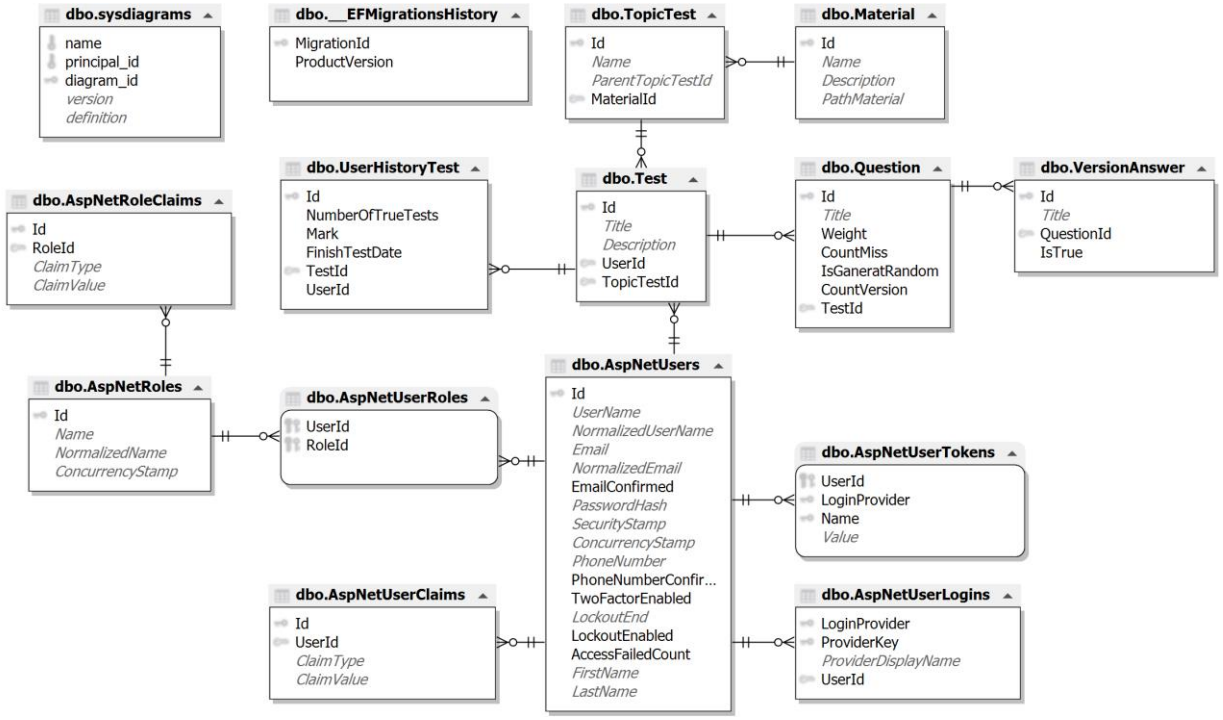


Рисунок 8. Спрощена ЕР-діаграма бази даних

Поле Email представляє адресу електронної пошти користувача. Поштова скринька є унікальною для кожного користувача системи. Як правило, поштові сервіси вимагають від нових користувачів підтвердження за номером телефону. В результаті збільшується захищеність системи, оскільки користувач не має можливості безконтрольно створювати будь-яку кількість несправжній користувачів для захаращення бази даних системи. Також дана адреса використовуєтсья для поштових нотифікацій з серверної частини системи.

Поле EmailConfirmed представляє інформацію про стан підтвердження поштової скриньки. Система має механізм перевірки належності поштової адреси користувачеві, аби виключити ситуації, коли користувач намагається використати чужу адресу для створення акаунту.

Поле PasswordHash представляє хеш паролю. Збереження паролів у відкритому вигляді у будь-яких, включаючи найзахищеніші бази даних є серйозною дірою у безпеці. Оскільки при отриманні доступу до бази несанкціонованої особи, або особи зловживаючою власним доступом до бази,

всі паролі всіх користувачів стають доступними. Зачасту користувачі використовують однакові паролі для різних сервісів, тому окрім загрози безпеки даної системи, такий підхід ставить під загрозу інші дані необачних користувачів. Тому в даній системі використано механізм хешування паролю. За допомогою функції незворотнього хешування пароль перетворюється в закодовану послідовність, з якої вже не можна отримати початкових даних. Таким чином для перевірки правильності паролю при автентифікації користувача система порівнює ввредений користувачем пароль, закодований тією ж функцією окремо з тим, що зребежено у базі даних.

Поле FirstName представляє ім’я користувача, а поле LastName представляє його прізвище. Ім’я та прізвище користувача зберігаютсья окремо. Вони використовуютсья для формування повідомлень та відображення в інтерфейсі користувача.

Поле PhoneNumber представляє номер телефону користувача використовується для додаткового рівня захисту системи по аналогії з механізмом поштової скриньки. Також він може бути використаний для зв’язку студента з користувачем системи та навпаки.

Поле PhoneConfirmed представляє інформацію про стан підтвердження номеру телефону. Як і у випадку з поштовою скринькою, система має механізм перевірки належності номеру телефону користувачеві для виключення ситуацій використання не належного йому номеру.

Поле AccessFailedCount представляє інформацію про кількість невдалих спроб авторизації, використовується для виявлення спроб взлому та блокування частоти спроб входу для протидії взлому. Також відбувається інформування каристувача про невдалі спроби входу.

Поле TwoFactorEnabled визначає, чи увімкнено для даного користувача режим двофакторної аутентифікації.

Поля LockoutEnabled і LockoutEnd визначають, чи доступ до акаунту користувача можна заблокувати і чи заблоковано його відповідно. Дані

розділені на два поля для виключення можливості помилки аміністратора системи при обмеженні доступу. Наприклад, аби не заблокувати свій аккаунт, або невідповідний аккаунт.

Поле SecurityStamp використовується для збереження випадкового значення, яке має змінюватися щоразу, коли змінюються облікові дані користувача для убезпечення від несанкціонованих змін даних користувача.

Поле ConcurrencyStamp використовується для запобігання конфлікту одночасного оновлення.

Таблиця AspNetRoles представляє сутність ролі користувача системи. Ключем є поле ідентифікатора.

Поле Name представляє назву ролі для відображення у інтерфейсі.

Поле NormalizedName як і відповідне поле таблиці AspNetUsers використовується для покращення швидкодії системи.

Поле ConcurrencyStamp так як і у таблиці AspNetUsers виступає в якості допоміжного засобу для вирішення конфлікту одночасного оновлення.

Таблиця AspNetUserRoles є розв’язуючою таблицею для зв’язку багато до багатьох між сутностями користувача та ролі, тобто відповідних таблиць AspNetUsers та AspNetUserRoles. Має складний ключ, який складається з двох полів UserId та RoleId, що зберігають ідентифікатори об’єктів відповідних сутностей.

Таблиця AspNetRoleClaims представляє сутність права ролі користувача системи. Ключем є поле ідентифікатора. Має посилання на таблицю AspNetUserRoles що зберігаєтсья в полі RoleId.

Поле ClaimType визначає тип доступного користувачеві, що має роль з відповідним правом, ресурсу.

Поле ClaimValue представляє найменування доступного користувачеві, що має роль з відповідним правом, ресурсу.

Таблиця AspNetUserClaims представляє сутність характеристики користувача системи. Ключем є поле ідентифікатора. Має посилання на таблицю AspNetUsers що зберігаєтсья в полі UserId.

Поле ClaimType визначає характеристику користувача. Поле ClaimValue представляє значення характеристики користувача.

Таблиця AspNetUserTokens представляє сутність JWT, присвоєного користувачеві. Має складний ключ, що складаєтсья з трьох полів: посилання на таблицю AspNetUsers, що зберігаєтсья у полі UserId; поля LoginProvider, що визначає спосіб підключення клієнту користувача до системи та поля Name, яке зберігає назву екземпляру токену. Поле Value представляє власне токен.

Таблиця AspNetUserLogin використовується для логування авторизацій користувача. Має складний ключ, що складаєтсья з двох полів: LoginProvider, що визначає спосіб підключення клієнту користувача до системи та поля ProviderKey, яке зберігає значення ключа підключення. Має посилання на таблицю AspNetUsers що зберігаєтсья в полі UserId.

Поле ProviderDisplayName використовуєтсья для збереження назви способу авторизації, зрозумілої для людини.

Таблиця Test представляє сутність тесту. Ключем є поле ідентифікатора. Має посилання на таблицю AspNetUsers що зберігаєтсья в полі UserId та на таблицю TopicTest, що зберігаєтсья в полі TopicTestId.

Поле Title представляє заголовок тесту. Поле Description представляє його опис.

Таблиця TopicTest представляє сутність теми тесту. Тема тесту виоремлена через те, що кожен тест може мати кілька тем. Ключем є поле ідентифікатора. Має посилання на таблицю Material що зберігаєтсья в полі MaterialId. Поле Name представляє назву теми. Також в таблись пристуній рекурсивний зв’язок. Таблиця посилається сама на себе за допомогою ідентифікатора, що зберігається в полі ParentTopicTestId. Таким чином досягається можливість збереження деревовидної структури у рамках таблиці.

Відповідно, запис в таблиці, в яккому не визначено значення ParentTopicTestId є корнем, всі решта записів є вузлами дерева і повинні включати посилання на батьківський вузол.

Таблиця Material представляє сутність довідки по темі. Ключем є поле ідентифікатора. Має посилання на таблицю TopicTest що зберігаєтсья в полі TopicTestId. Поле Name представляє назву матеріалу. Поле Description представляє коротку викладку матеріалу теми тесту. Поле PathMaterial представляє посилання на детальну версію матеріалу по темі.

Таблиця UserHistoryTest призначенна для збереження результатів проходження тестів. Ключем є поле ідентифікатора. Має посилання на таблицю Test що зберігаєтсья в полі TestId та посилання на таблицю User що зберігається у полі UserId. Поле NumberOfTrueTests представляє кількість успішно складених тестів. Поле Mark слугує для збереження оцінки, отриманої за складений тест. Поле FinishTestDate слугує для збереження дати проходження тесту і подальшого сортування вибірки.

Таблиця Question представляє сутність питання. Ключем є поле ідентифікатора. Має посилання на таблицю Test що зберігаєтсья в полі TestId. Поле Title представляє власне питання. Поле Weight слугує збереження для поточної ваги динамічного показника значимості тесту. Поле CountMiss представляє кількість неправильних відповідей на дане питання в межах проходження тесту користувачами системи. Поле IsGeneratRandom визначає чи відбувається випадкові вибірка варіантів правильних відповідей на дане питання з відповідної таблиці при запиті питання. Поле CountMiss представляє кількість неправильних відповідей на дане питання в межах проходження тесту користувачами системи. Поле CountVersion визначає скільки варіантів відповіді буде присутньо у вибірці при запиті питання.

Таблиця VersionAnswer представляє сутність варіанту відповіді. Має посилання на таблицю Question що зберігаєтсья в полі QuestionId. Ключем є

поле ідентифікатора. Поле Title слугує для збереження значення варіанту відповіді. Поле IsTrue вказує чи є варіант правильним.

Таблиці EFMigrationsHistory та sysdiagrams є службовими. Таблиця EFMigrationsHistory слугує для зебереження даних про міграції бази даних. Таблиця sysdiagrams використовуєть для жбереження інформації про діаграми бази даних.

Таким чином розроблена база даних повністю задовольняє функціональні вимоги розроблюваної системи тестування та представляє можливість консистентного збереження та вибірки даних. В розробленій базі даних відсутні аномалії вибірки та збереження даних.

3.2 Компоненти клієнтської частини системи та їх взаємодія

3.2.1 Структура клієнтського додатку на базі фреймворку Angular

Angular - це платформа та фреймворк для побудови клієнтських додатків. Він реалізує основну та додатковий функціонал як набір бібліотек TypeScript, які імпортуються у програми.

Основними складовими частин програми Angular є NgModules, які забезпечують контекст компіляції компонентів. NgModules збирають відповідний код у функціональні набори; додаток Angular визначається набором NgModules. У додатку завжди є принаймні кореневий модуль, який є точкою входу, і, як правило, має ще багато функціональних модулів.

Компоненти (Components) визначають представлення даних, що являють собою набори елементів екрану, які можуть оброблятися та змінюватися за допомогою Angular відповідно до логіки та даних програми.

Компоненти використовують сервіси (Services), які надають певні функціональні можливості, безпосередньо не пов'язані з представленням. Постачальники сервісів можуть бути введені в компоненти як залежності, що забезпечує модульність, багаторазовість та ефективність коду.

І компоненти, і сервіси - це класи, в яких є декоратори, які позначають їх тип та надають метадані (Metadata), які вказують Angular, як їх використовувати.

Метадані класу компонентів асоціюють його з шаблоном (Template), який визначає представлення. Шаблон поєднує звичайний HTML з директивами (Directives) Angular та розміткою, що дозволяють фреймворку змінювати HTML, перш ніж відобразити для представлення.

Метадані класу обслуговування надають інформацію, яку Angular потребує, щоб зробити його доступним для компонентів через механізм Dependency Injection (DI).

Компоненти програми зазвичай визначають багато представлень, розташованих ієрархічно. Angular надає послугу маршрутизатора, для визначення шляхів навігації серед представлень. Маршрутизатор забезпечує складні навігаційні можливості в браузері.

Схема представлена на рисунку 9 ілюструє дані відношення.

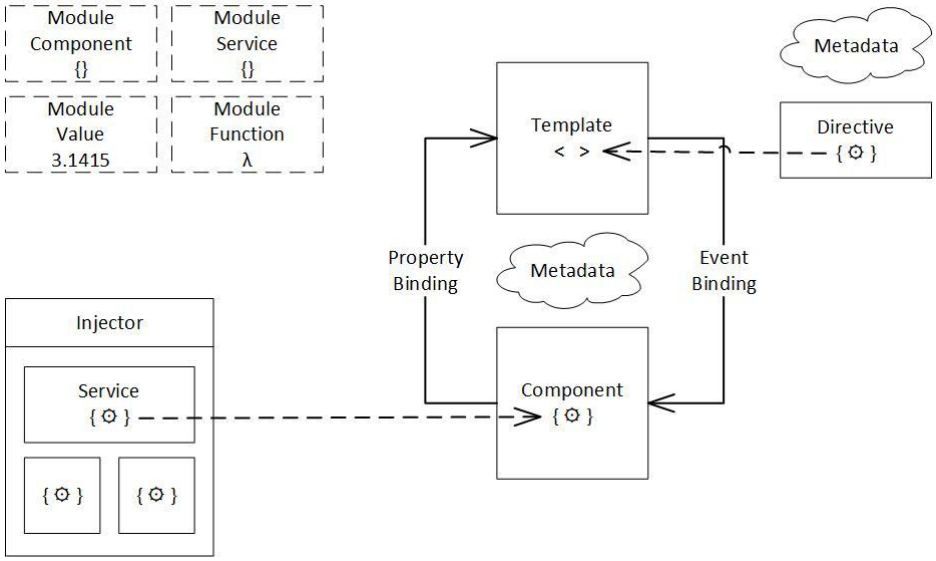


Рисунок 9. – Схема архітектурних компонентів клієнтського додатку

3.2.2 Огляд модулів реалізованого клієнтського додатку

У кінцевому додатку виділено наступні модулі:

- AppModule – основний модуль, точка входу та конфігурація роботи додатку;
- ApiModule – модуль, який інкапсулює логіку зв’язку з серверною частиною системи;
- AuthModule – модуль, що інкапсулює логіку представлення та обробки даних користувача, авторизації автентифікації та реєстрації;
- AdminModule - модуль, що інкапсулює логіку представлення та обробки даних адміністратора, механізми редагування тестів, адміністрування системи;
- TestModule – модуль, який інкапсулює логіку проходження тестів користувачем. Відображення і функціонал тесту та його елементів;
- SharedModule – модуль, що включає спільна компоненти інших модулів, UI директиви та компоненти позбавлені логіки доступу до сервісів даних;
- AuthModule – модуль, що інкапсулює логіку JWT автентифікації та авторизації користувача;
- DashboardModule – модуль, що включає логіку відображення акаунту користувача, історій пройдених тестів;
- ToastModule – модуль, що реалізує функціонал вспливаючих повідомлень на нотифікайій про помилки;
- LocalStorageModule – модуль, що включає логіку збереження типізованих даних у локальному сховищі клієнтського додатку;
- ThemingModule – модуль, що включає логіку збереження типізованих даних у локальному сховищі клієнтського додатку.

3.3 Контроль доступу

У системі тестування передбачено наступні ролі користувачів:

- власне користувач – має права на проходження тестів. Редагування власної інформації, перегляд навчальних матеріалів;
- адміністратор – має доступ до редактора тестів. Може створювати та конфігурувати нові тести.

Для вирішення проблеми контролю доступу було імплементовано автентифікацію на базі Json Web Token (JWT) виходячи з наступних перевах технології:

- головна перевага: оскільки метод ніяк не оперує станами, сервер не вимушений зберігати записи до призначених для користувача токенів або сесій. Кожен токен самодостатній, містить всі необхідні для перевірки дані, а також передає затребувану призначену для користувача інформацію. Тому токени не ускладнювати масштабування;
- при використанні Cookies відбувається збереження лише ID сесії користувачів, а JWT дозволяє зберігати метадані будь-якого типу, якщо це коректний JSON;
- при використанні Cookies сервер повинен виконувати пошук по традиційній SQL-базі або NoSQL-альтернативі, і обмін даними триває довше, ніж розшифровка токену. Крім того, JWT дозволяє заощадити додаткові звернення на пошукові запити для отримання та обробки даних користувача;
- при використання Cookies на мобільних платформах є багато обмежень і особливостей. JWT-підхід набагато простіше реалізувати на iOS і Android, а також для додатків і сервісів, в яких не передбачено зберігання Cookies;
- при аутентифікації на основі токенів не відстежуються стани. Не відбувається збереження інформації про користувачів на серверах або в сесіях і навіть не зберігаєтсья сам JWT, використаний для клієнтів.

Процедура аутентифікації на основі токенів може представлена наступною послідовністю дій:

а) користувач вводить ім'я та пароль;

b) сервер перевіряє їх і повертає токен (JWT), який може містити такі метадані як ідентифікатор користувача, ролі, тощо;

c) токен зберігається на клієнтській стороні, частіше всього в локальному сховищі;

d) наступні запити до серверів зазвичай містять цей токен в якості додаткового заголовку авторизації у вигляді «Bearer {JWT}». Також токен може пересилатися у тілі POST-запиту і як параметр;

e) сервер розшифровує JWT, якщо токен валідний, сервер обробляє запит.

Коли користувач виходить з системи, токен на клієнтській стороні знищується, з взаємодія з серверами не потрібна. Дану послідовність представлено на рисунку 10.

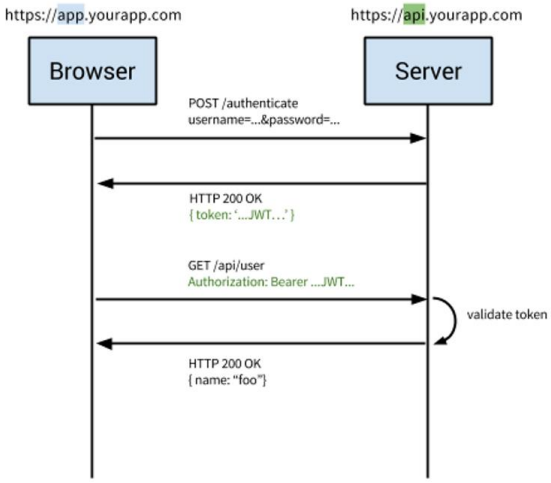


Рисунок 10. Діаграма послідовності аутентифікації за допомогою JWT

3.4 Інтерфейс користувача

UX і UI - головні інструменти створення дизайну. Так як сьогодні для поліпшення користувацького досвіду направлена вся увага дизайнера, продукт треба робити максимально зручним і зрозумілим для користувача. Саме UX/UI найкраще підходять для досягнення цієї мети. З їх допомогою сайт можна зробити красивим і зручним з чіткою структурою.

Розвиток напряму UX/UI породив тренди, які впливають на дизайн:

- спрощення візуальної частини інтерфейсів. Дизайнеру тепер не потрібно витрачати час на відмалювання рельєфних елементів інтерфейсу і пошук добре прочитуваних образів. Сучасний користувач легко розуміє умовні позначення. Вже не потрібно пояснювати, наприклад, що кольоровий прямокутник - це кнопка, а три горизонтальні смужки - меню.
- ускладнення самого дизайну, як дисципліни. Інтерфейси спростилися, але сам дизайн став складнішим і багатошаровим. Зараз сайти представляють собою комплексні системи з безліччю різних елементів: формами заявки, особистим кабінетом, тощо. Потрібно чітко продумувати шлях користувача на сайті, щоб він не заблукав, швидко знаходив потрібні розділи, а в ідеалі - здійснював цільові дії.

Користувачі повинні відразу зрозуміти, де вони і куди треба клікнути, щоб переглянути історію тестів, прочитати статтю, пройти тест і так далі.

Тут важливі інтерфейс, кольори, графіка і шрифт - все це допомагає виділити важливе.

Згідно зі статистикою, на мобільні пристрої доводиться 5,135 млрд людей - на 4% більше, ніж рік тому, і це число зростає. [15] Роблячи адаптивні сайти, відбувається орієнтація більш ніж на половину аудиторії.

Коректне відображення сайту на мобільних пристроях – вимога сучасного дизайну.

Крім того, пошукова система Google спочатку індексує сайти з мобільною версією і понижує у видачі старі неадаптивні.

Також має зручну навігацію, що відповідає зваженій логічній структурі. Як і очікують користувачі, Користувачі очікують, панель навігації перебуває на лівій бічній панелі.

URL-адреси допомагають відвідувачам визначити організацію сайту і зрозуміти, який контент можна очікувати на сторінці. URL-адреси сервісу

складаються з ключових слів, які спроектовані так, що мають сенс в першу чергу для людей, а не для пошукових систем.

3.5 Тестування та якість коду

Повільне завантаження веб-сторінок є основним дратівливим чинником при роботі з сервісом. Малоймовірно, що користувачі чекатимуть оновлення сторінки більше кількох секунд. Тому система повинна задовільняти вимогу мінімального часу завантаження.

Згідно досліджень Kissmetrics, односекундна затримка може скоротити конверсію на 7%. До 25% покидають сторінки з часом завантаження в чотири секунди. [15] Дані представлені на рисунку 11.

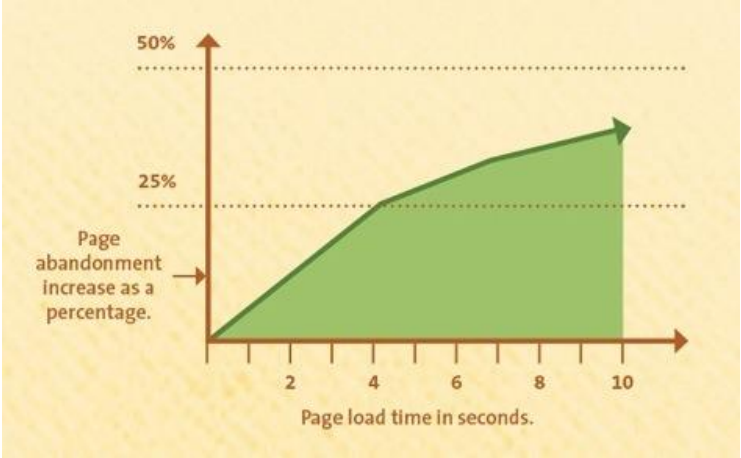


Рисунок 11. Графік відношення вірогідності покидання сайту користувачем до часу завантаження сторінки

Для тестування швидкодії сервісу, виявлення «вузьких місць» та основних проблем клієнтської частини використано ресурс Google PageSpeed Insights. Результати тестування представлено у додатку В. Як видно з приведених у додатку даних, розроблена система повністю відповідає сучасним вимогам по швидкодії та доступності ресурсів і отримала високу загальну оцінку роботи.

Також для перевірки функціонування сайту, регулярно, після впровадження нового функціоналу, проводилося мануальне тестування основних зачеплених модулів.

Крім цього проведено тестування коректного відображення клієнту на різних пристроях шляхом симуляції відповідних характеристик екрану та фізичного інтерфейсу за допомогою інструментів розробника Google DevTools. Іллюстрацію тестування приведено на рисунку 12.

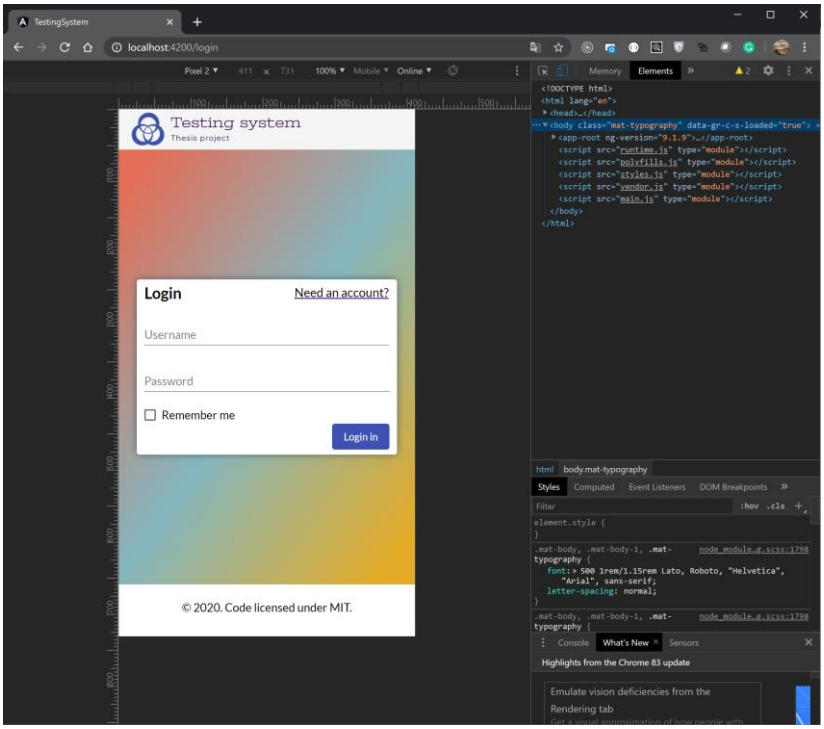


Рисунок 12. – симуляція запуску клієнту на пристрої Pixel 2

Лінтер (Linter) – це приладдя для дослідження програмного коду з метою виявлення помилок програмування, вад, порушень стилю, а також сумнівних чи підозрілих виразів

Для виконання вимог якості коду та підтримуваності системи вирішено застосувати наступні утиліти, а саме лінтери:

- TSLint – для перевірки коду мовою TypeScript;
- SCSSLint – для перевірки каскадних таблиць стилів;
- HTMLLint – для перевірки шаблонів представлень.

3.6 Реалізація функції динамічної ваги правильної відповіді

Для реалізації функції динамічної ваги оцінки правильної відповіді на питання тесту, відповідно до розробленого технічного завдання було розроблено механізм перерозподілу ваги відповідей на питання на основі побудови відношення оцінки кожного окремого питання в тесті, що враховує кількість правильних та неправильноих відповідей на нього користувачами.

Даний механізм працює наступним чином: по завершенню проходження окремо взятого тесту кожним з користувачів, серверна частина застосунку асинхронно виконує перерозподіл вагового балу кожного питання, базуючись на оновленні статистичної вибірки, а саме зміні значень кількості правильних та не правильних відповідей на окремо взяте питання рештою користувачів, що пройшли даний тест. Для встановлення ваги окремо взятого питання, в відповідності до описаного алгоритму, маємо наступну формулу:

$$w = 100\frac{Q_m}{\sum_0^Q Q_m},$$

де w – це вага відповіді, Q_m – кількість неправильних відповідей для одного запитання, Q – кількість питань в тесті.

При встановленні значення ваги правильної відповіді попередньою формулою є ризик того, що вага питання може прямувати до нуля, що не допустимо при застосуванні в рамках оцінювання. Як ррішення проблеми було розроблено обмеження, що регулюється формулою:

$$w_{min} = \frac{100}{\sqrt{Q^3}},$$

де w_{min} – мінімальна вага правильної відповіді, Q – кількість питань в одному тесті.

ВИСНОВКИ

У результаті виконання дипломного проекту було спроектовано індивідуальну частину системи автоматизованого тестування на основі відносного оцінювання по статистичним вибіркам, розроблено механізми міжмодульної взаємодії та реалізовано поведінку відповідних модулів.

Було проведено аналіз предметної області, в результаті чого визначено основні вимоги до формату тестів, що генеруються системою. У ході огляду існуючих рішень було розглянуто найпопулярніші веб-орієнтовані системи тестування, визначено їх переваги та недоліки, встановлено вимоги до клієнтської частини. Як архітектурне рішення для побудови системи обрано клієнт-серверний додаток. Як технічне рішення для клаєнтської частини, у ході аналізу технологій розробки клієнтських додатків, було вирішено побудувати односторінковий додаток на базі фреймворку Angular мовою TypeScript з використанням препроцессору каскадних таблиць стилів SCSS та розмітки мовою HTML. Як СУБД, у ході огляду технологій збереження даних, було вирішено використати Microsoft SQL Server.

Виконано проектування бази даних. Розроблено ключові таблиці, каскади для збереження тестів, користувачів системи. Описано компоненти системи та їх взаємодію. Для вирішення завдання аутентифікації у межах системи було розроблено механізм аутентифікації за допомогою JWT-токену. Розроблено формат взаємодії клієнтської та серверної частини зі сторони клієнту, вимоги до інтерфейсу користувача та концепцію його зовнішнього вигляду. На основі сформованого технічного завдання проведено синтез системи. Складено план мануального тестування системи та тестування з використанням сторонніх ресурсів. Визначено вимоги до швидкодії та відмовостійкості системи. Реалізовано основні компоненти та функціонал системи. На базі плану тестування виконано відлагодження системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буйницька О. П., Інформаційні технології та технічні засоби навчання. - Київ: Центр навчальної літератури, 2017.
2. Why do we need an Online Exam System? [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://fedena.com/blog/2019/08/online-exam-system.html (дата звернення: 10.04.2020). – Назва з екрана.
3. Exam software. - URL: https://www.capterra.com/exam-software (дата звернення: 11.04.2020). – Назва з екрана.
4. Використання тестових систем. [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://pidruchniki.com/10611207/informatika/vikoristannya_testovih_sistem (дата звернення 15.04.2020) – Назва з екрана.
5. Len Bass. Software Architecture in Practice (SEI Series in Software Engineering) / Paul Clements, Rick Kazman - Addison-Wesley Professional, 3 видання, 2012.
6. Andrew S. Tanenbaum. Computer Networks. / David J. Wetherall. - Prentice Hall. 5th Edition. 2010.
7. Bartosz Szczeciński. What’s Server Side Rendering and do I need it? [Електронний ресурс]:[Веб-сайт]. – Режим доступу: https://medium.com/@baphemot/whats-server-side-rendering-and-do-i-need-it-cb42dc059b38 (дата звернення 25.04.2020). – Назва з екрана.
8. Thick Client Definition. [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://techterms.com/definition/thickclient (дата звернення: 27.04.2020).
9. Thin Client Definition. [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://techterms.com/definition/thinclient (дата звернення: 27.04.2020). – Назва з екрана.
10. Pautasso Cesare. REST: Advanced Research Topics and Practical Applications. / Wilde Erik, Alarcon Rosa. - Springer, 2014.

11. Cody Lindley. Front-end Developer Handbook, 2019. [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://frontendmasters.com/books/front-end-handbook/2019/ (дата звернення: 14.05.2020) – Назва з екрана.

12. C. J. Date, "SQL and Relational Theory: How to Write Accurate SQL Code 2nd edition". - O'reilly Media, Inc., 2012.

13. Abandoning ACID in Favor of BASE in Database Engineering. [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674 (дата звернення: 29.04.2020) – Назва з екрана.

14. Jeffrey Dean. MapReduce: Simplified Data Processing on Large Clusters. / Sanjay Ghemawat [Електронний ресурс]:[Веб-сайт]. – Режим доступу http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf (дата звернення: 29.04.2020). – Назва з екрана.

15. Brian Sutter. Website Speed Matters. [Електронний ресурс]:[Веб-сайт]. – Режим доступу https://www.entrepreneur.com/article/281986 (дата звернення: 17.05.2020). – Назва з екрана.

ДОДАТОК А

Публікація на тему роботи на конференції Summer Infocom 2020

Система тестування на основі відносного оцінювання по статистичним вибіркам

Коваленко Ігор
КПІ ім. Ігоря Сікорського
Київ, Україна
meelistenso@gmail.com

Вовчок Євген
КПІ ім. Ігоря Сікорського
Київ, Україна
vovchok.zhenya@gmail.com

Анотація. **Мета публікації – переглянути наявні способи вирішення завдання по проведенню оцінювання якості знання способом тестування і змоделювати архітектурне рішення, яке поєднує всі переваги, а також не буде містити недоліків оглянутих аналогів. Оптимальним рішенням в межах обумовлених в статті критеріїв є програмний продукт створений на основі клієнт – серверної архітектури в якій серверний компонент реалізовано у вигляді єдиної архітектурної одиниці, для забезпечення простоти розгортання та зниження вартості підтримки системи**

Ключові слова: системи тестування; встановлення рівня знань; статистична вибірка; клієнт–серверна архітектура.

У сучасних реаліях ні одна сфера життя не обходиться без оцінки якості рівня знань та підготовки спеціалістів по тим чи іншим напрямам. У результаті постає завдання якісної та коректної оцінки знань, з можливістю їх подальшого вдосконалення в межах процесу навчання, який обумовлено процесом виявлення відсутності знань в межах певного представленого графу знання, по тій чи іншій спеціальності. Дане завдання в сучасному світі не може бути не вирішеним, проте розглядаючи існуючі рішення та наявні життєві приклади можна говорити про недосконаліть, неточність та нешаблонність даних рішень.

Після огляду популярних систем тестування, представлених на веб-порталах, зокрема, мережевої академії Cisco, Canvas, Blackboard Learn, Kahoot!, та на основі досвіду проходження тестів під час навчання, було сформовано перелік основних типів тестів. Для повної реалізації функціональних вимог і конкурентоздатності з переліку було обрано наступні види тестів, які повинні міститися у системі:

1) Правильно/неправильно – користувач повинен визначити чи правдиве твердження в питанні. Це найпростіший тип питання.
2) Вибір однієї відповіді – користувачу потрібно вибрати одну правильну відповідь з запропонованих варіантів.
3) Вибір кількох відповідей – користувач обирає правильні варіанти зі списку. Завдання такого типу складніші чим вибір однієї відповіді, так як кількість правильних відповідей заздалегідь невідома. Правильна відповідь методом випадкового підбору малоймовірна.
4) Коротка відповідь – користувач повинен ввести правильну відповідь в текстове поле. Для правильної відповіді необхідно добре розбиратись у тематиці поточного питання.
5) Послідовність – користувач розміщує елементи у правильній послідовності.
6) Числова відповідь – користувач вводить число у поле для відповіді. Вгадати правильну відповідь малоймовірно.
7) Відповідність – користувач повинен з'єднати пари слів, фраз, або зображень. Додаткові «зайві» варіанти відповідності можуть ускладнити питання.

При створенні тесту є необхідність виставлення прохідного балу. Крім того, існує проблема встановлення еквівалентності ваги правильних відповідей, оскільки різні питання можуть посилатися на різні об'єми знань, можуть мати неточності, бути складними для розуміння. Універсальний рецепт для фіксованого значення прохідного балу відсутній, тому необхідно зробити його відносним. Як рішення для забезпечення об'єктивності тестування обрано динамічне визначення ваги правильних відповідей відносно загальної статистики відповідей користувачів на питання тесту у межах системи.

Для того, щоб охопити якнайбільшу область знань, але при цьому не виснажувати користувача, понижуючи його продуктивність було вирішено надати можливість встановлення довільної кількості запитань адмінінстратором системи та встановлено рекомендований обсяг 25-30 питань. При цьому загальний банк питань повинен бути не менше ніж у 3-4 рази більшим, ніж кількість питань у згенерованому адміністратором тесті з метою забезпечення унікальності тестів у різних користувачів і виключення можливості плагіату та розповсюдження набору готових питань-відповідей на тест. Також для виключення можливості користування додатковими матеріалами користувачем необхідно, щоб тест передбачав обмеження по часу, яке встановлюється адміністратором.

На сьогоднішній день існує безліч систем онлайн-тестування. Скориставшись пошуковою системою Google, та переглянувши знайдені веб-

ресурси і рейтинги, можна знайти ряд схожих за функціоналом та призначенням систем.

Відповідно до призначення тести поділяються на:

1) Навчальні – ті, що допомагають закріпити вивчений матеріал. Зазвичай такі тести можна знайти після кожної глави у курсі в якості невеликої практики. Відсутні обмеження по часу, штрафи за неправильні відповіді. На вирішення задач надається кілька спроб. Після кожної помилки відображаються пояснення. Як приклад, тести що стають доступні після завершення вивчення глав у системі мережевої академії Cisco.

2) Атестаційні – допомагають оцінити знання користувача. В таких тестах присутні обмеження по часу, дається одна спроба на відповідь, відсутні пояснення помилок. Як приклад, Exam.net.

За спеціалізацією, тестові системи поділяються на:

1) Спеціальні – ті, що призначені для застосування лише у певних сферах. Як приклад, нині популярні системи тестування персоналу SHL, TalentQ, Kenexa, або система тестування інтегрована у платформу Udemy, призначена для оцінки засвоєних навичок після проходження курсу.

2) Загальні – ті, що мають широке призначення. Різноманітні конструктори тестів та більш загальні системи як Google Forms.

Більшість наявних систем не об'єднують тестування та навчання шляхом встановлення незнання та його ліквідацію за допомогою опису і вказання правильних відповідей шляхом пояснень. Крім того ні одна з них не вирішує проблему еквівалентності ваги правильних відповідей. Також присутня проблема різноманітності форми тестових завдань і сучасності UX [1] частини систем. Як приклад Moodle, мережева академія Cisco мають застарілий дизайн та вузький перелік форм тестових питань.

1) У результаті огляду прийнято наступні вимоги до проектованого рішення:

2) Можливість додавання пояснень та посилань до відповідних питанням тем та їх пов'язування.

3) Вирішення проблеми еквівалентності ваги правильних відповідей шляхом впровадження динамічного показника відносного загальній статистиці відповідей на питання тесту.

4) Впровадження різноманітних форм тестових питань з можливістю конфігурації.

5) Розробка сучасного дизайну.

6) Широка спеціалізація системи, шляхом налаштувань формату тестів.

Відповідно до сформованих вимог про наявність багатьох користувачів та необхідність централізованого збереження інформації про результати тестів, їх оцінювання, для даного класу підсистем доцільне використання наступних архітектур міжкомпонентної взаємодії: SOA, client-server [2].

Клієнт-серверний підхід

Даний шаблон складається з двох частин: сервера і безлічі клієнтів, рис 1. Серверний компонент надає сервіс клієнтським компонентам. Клієнти запитують сервіс у сервера, а він, у свою чергу, надає їх клієнтам [3].

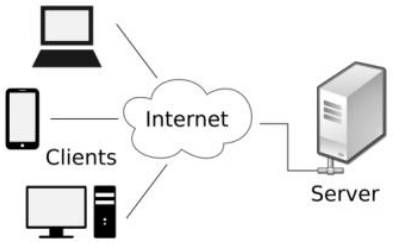


Рис. 1. Схема комп'ютерної мережі клієнтів, що спілкуються з сервером через мережу Інтернет

Зазвичай використовується для реалізації онлайн додатків (електронна пошта, спільний доступ до документів, банківські послуги, тощо).

Переваги:

1) Підходить для моделювання набору сервісів, які можуть запитувати клієнти.

Недоліки підходу:

1) Запити зазвичай виконуються в окремих потоках на сервері.

2) Взаємодія між процесами підвищує ресурсовитратність, тому що різні клієнти мають різне представлення.

Підхід SOA

Сервісно-орієнтована архітектура (SOA) – спосіб побудови міжкомпонентної взаємодії програмного забезпечення, за якого послуги надаються іншим компонентам компонентами додатків, через протокол зв'язку по мережі [3]. Сервіс SOA - це дискретна одиниця функціональності, до якої можна отримати доступ дистанційно, діяти та оновлюватись незалежно, наприклад, отримання виписки з кредитної картки в Інтернеті.

В межах SOA архітектури виділяють наступні елементи, які відображені на рис. 2.

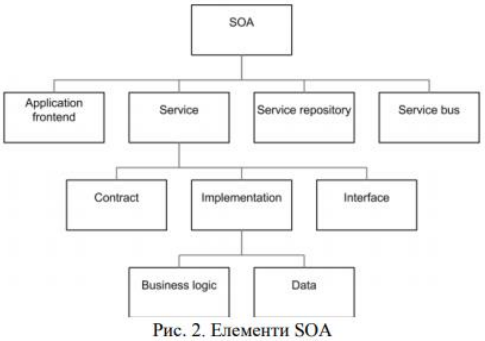


Рис. 2. Елементи SOA

Сервіс відповідно до визначення SOA, має наступні властивості:

1) Представлення поведінкової логіки із заданим результатом.
2) Обмежена відповідальність в межах одного бізнес-завдання.
3) Це чорна скринька для споживачів, тобто споживач не повинен знати про внутрішню роботу служби.
4) Можливість об'єднання набору інших сервісів.

Різні сервіси можуть бути використані спільно, для забезпечення функціональності складної логіки поведінки по комплексній обробці користувацького запиту, що потребує поєднання окремо взятої логічної поведінки в межах єдиної відповідальності кожного сервісу для видачі результату. Проте даний підхід має ряд недоліків:

1) Неоднорідність і складність рішення.
2) Величезний набір тестових комбінацій завдяки інтеграції автономних сервісів.
3) Включення послуг від різних конкуруючих постачальників.
4) Платформа постійно змінюється через наявність нових функцій і послуг.

Технічне рішення

Під час розробки технічного рішення для серверної частини було розглянуто найпопулярніші технології для побудови веб застосунків а саме CGI (C++), ASP.NET CORE (C#) та Spring (Java) [1].

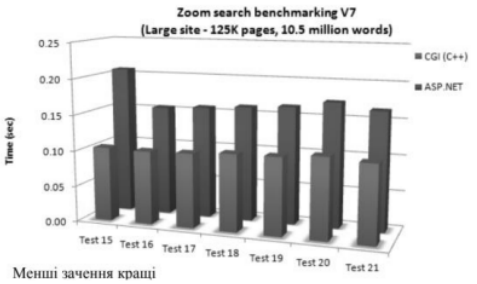


Рис. 3. Порівняння швидкодії реалізацій серверних рішень CGI та ASP.NET CORE

В першу чергу було розглянуто порівняльну характеристику CGI та ASP.NET CORE, а саме швидкість пошуку та видачі даних на порівнюваних платформах. Виходячи з даних графіку представленого на рис. 3, рішення на базі CGI має більшу швидкодію. Проте, на відносно невеликих об'ємах даних розрив у швидкодії скорочується. Крім цього розробка на базі мови C++ є значно витратнішою та займає більше часу, через технічні особливості мови та наявність безпосереднього управління оперативною пам'яттю в межах створюваного процесу [7].

В межах подальшого огляду було проведено порівняння ряду конфігурацій систем ASP.NET CORE та системи на базі Spring [8], результати якого представлено на рис. 4.

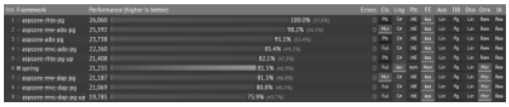


Рис. 4. Порівняння швидкодії різних конфігурацій систем на базі ASP.NET CORE та системи на базі Spring

Виходячи з результатів тестів та порівнянь, були зроблені висновки, що для розробки об'ємного проекту найкраще підходить технологія ASP.NET CORE. Протягом вирішення завдання по збереженню та взаємодії з даними, до порівняння було взято відомі СУБД, а саме: PostgreSQL, MS SQL і IBM db2.

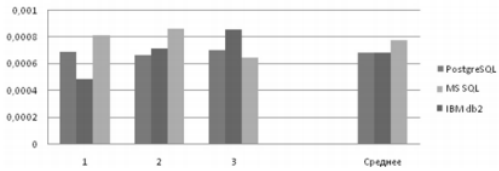


Рис.5. Порівняння швидкодії реалізацій SQL серверних запитів

На основі даних, представлених на рис. 5, було виявлено перевагу MS SQL у швидкості реалізації серверних запитів порівняно з рештою розглянутих СУБД. Виходячи з обраних технологій, постало питання прийняття рішення по вибору способу взаємодії між сервером і базою даних. Оптимальним вибором для прийнятого стеку технологій є Entity Framework [5] на основі наступних критеріїв:

1) Наявність підтримки способів генерації бази даних CodeFirst, DataBaseFirst.
2) Наявність можливості написання запитів засобами мови LINQ та їх подальше виконання всередині бази даних SQL.
3) Містить генерацію бази даних з відповідно заданої об'єктно-орієнтованої моделі.
4) Наявність механізму кешування даних.

Під час розробки технічного рішення для клієнтської частини було розглянуто найпопулярніші фреймворки односторінкових веб-додатків, а саме Vue.js, Angular та React.

Розглянувши результати порівняння швидкодії, представлені на рис. 6, було визначено, що всі перелічені фреймворки мають схожу продуктивність під час виконання програми. Виділяється лише швидкість першого завантаження сторінки з використанням Angular. Це пояснюється тим, що даний фреймворк включає в собі ряд бібліотек, призначених для організації коду, взаємодії з зовнішніми джерелами даних, маршрутизації всередині додатку, організації потоків даних. У випадку розробки веб-орієнтованого додатку зі складним інтерфейсом це є значною перевагою, оскільки решта фреймворків потребує власної імплементації даного функціоналу, що у подальшому значно впливає на

швидкодію продукту та ускладнює розробку системи.

Виходячи з цього для розробки клієнтської частини було обрано фреймворк Angular.

Name	vue-v2.5.16-keyed	angular-v6.0.0-keyed	react-v16.1.0-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	79.4 ±1.0 (1.0)	137.1 ±0.8 (1.7)	82.4 ±1.0 (1.0)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	17.0 ±0.5 (1.0)	55.9 ±0.5 (3.3)	20.7 ±0.4 (1.2)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	177.8 ±2.0 (1.0)	233.8 ±2.0 (1.3)	181.9 ±2.1 (1.0)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	232,029.0 ± 0.0 (1.0)	384,701.0 ± 0.0 (1.7)	266,195.0 ± 0.0 (1.1)

Рис. 6. Порівняння швидкодії фреймворків Angular, React та Vue.js

Власна реалізація

Під час програмної обробки в межах механізму оцінювання тесту робота якого обмежена специфікою роботи Web API та специфікацією HTTP протоколу виникають наступні технічні проблеми:

1) Перерахунок ваги кожного питання.
2) Оновлення спільних даних конкурентними потоками.

Був розроблений механізм перерозподілу ваги відповідей на питання, на основі побудови відношення оцінки кожного окремого питання в тесті, що враховує кількість правильних та неправильних відповідей на нього користувачами. Відповідно, по завершенню проходження окремо взятого тесту кожним з користувачів, серверна частина застосування в фоновому потоці виконує перерозподіл додаткового вагового балу кожного питання, базуючись на оновленні статистичної вибірки, а саме зміні значень кількості правильних та не правильних відповідей на окремо взяте питання рештою користувачів, що пройшли даний тест. Для встановлення ваги окремо взятого питання, в відповідності до описаного алгоритму, маємо наступну формулу:

$$w = \frac{100 Q_m}{\sum_0^Q Q_m},$$

де w – це вага за питання, Qm – це кількість не вірних відповідей для одного запитання, Q – це кількість питань в тесті. При встановленні значення ваги запитання попередньою формулою є ризик того, що вага питання може прямувати до нуля, що є критичним в нашій системі. Для рішення цього завдання було зроблено обмеження, яке корегується наступною формулою:

$$w_{\min} = \frac{100}{\sqrt{Q^3}},$$

де $w_{\min}$ – це мінімальна вага за питання, Q – це кількість питань в одному тесті.

Під час моделювання системи було виявлено наступну технічну проблему, обумовлену можливістю появи конкуруючих потоків, що в свою чергу будуть оновлювати спільні набори даних в один і той самий момент. З метою вирішення технічної проблеми було розроблено наступний механізм, що на основі планувальнику задач реалізує централізоване управління операціями зміни значень з точкою синхронізації, що створюється в межах додаткового джерела даних, на основі записів поточних та запланованих задач. В якості планувальника задач було використано Hangfire [8], як одне з найпопулярніших рішень в межах технології що використовується. Такий підхід забезпечує рішення технічної проблеми конкурентих потоків в межах всієї системи, що в свою чергу надає можливість зміни та збереження статистичних даних консистентно.

Висновки

В межах наявного огляду виділено клас підсистем направлених на встановлення оцінки якості рівня знань та підготовки спеціалістів по тим чи іншим напрямам. Визначено їх переваги та недоліки, на основі яких встановлено функціональні вимоги до даної системи. Виходячи з функціональних вимог було обрано, як архітектурне рішення клієнт-серверний додаток. Для реалізації архітектурного рішення та функціональних вимог було проведено огляд технічних засобів, що застосовують для створення даного класу систем і сформовано технічне рішення, яке включає побудову односторінкового додатку на основі фреймворку Angular на боці клієнтської сторони системи та додатку на основі платформи .NET Core та СУБД MS SQL Server з боку серверної частини. У даній системі реалізовано логіку динамічного визначення ваги правильних відповідей, яка забезпечує об'єктивність результатів тестів.

Література

[1] https://ela.kpi.ua/bitstream/123456789/27860/1/Vovk_magistr.pdf (дата звернення 12.05.2020).

[2] Леон Шкляр, Річ Розен., книга «Архітектура веб - додатків», рік видання: 2011 року

[3] https://echo.lviv.ua/dev/6455 (дата звернення 11.05.2020).

[4] https://www.ibm.com/developerworks/ru/library/ws-soa-term1/index.html (дата звернення 11.05.2020).

[5] https://habr.com/en/post/262461/ (дата звернення 12.05.2020).

[6] https://www.techempower.com/benchmarks (дата звернення 12.05.2020).

[7] https://docs.microsoft.com/en-us/ef/ef6/ (дата звернення 12.05.2020).

[8] https://www.hangfire.io/ (дата звернення 12.05.2020).

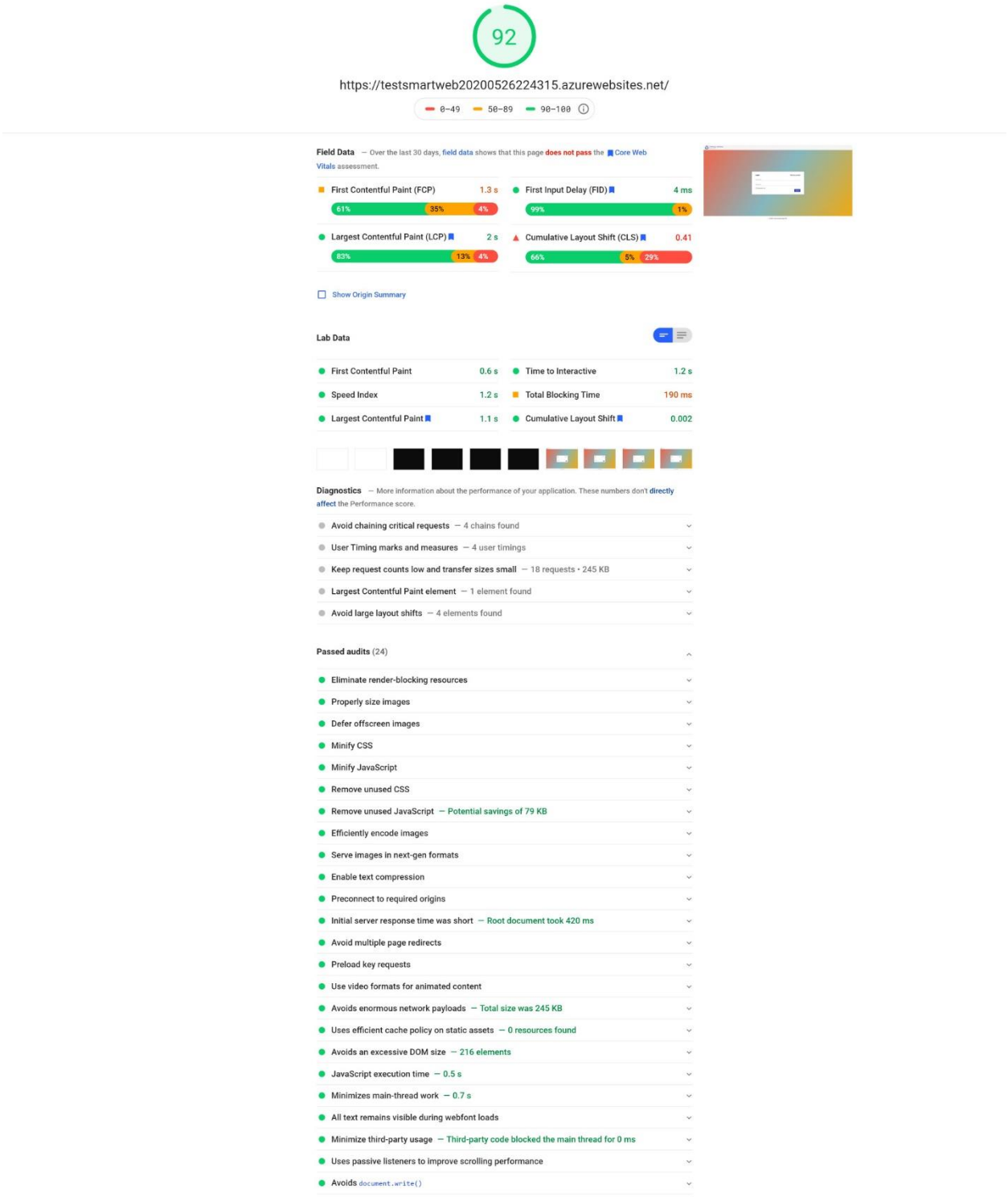
ДОДАТОК Б

Таблиця порівняння критичних факторів сучасних фронтенд-фреймворків

	Vue	React	Angular
Варіанти використання	Створення динамічних або односторінкових додатків	Створення динамічних або односторінкових додатків за допомогою React, та мобільних додатків з використанням React Native.	Прогресивні веб-додатки, гібридні мобільні додатки та веб-додатки корпоративного рівня.
Функціонал	Обмежена основна функціональність, яку можна легко розширити за допомогою сторонніх рішень	Бібліотека з обмеженою функціональністю, яку можна розширити за допомогою сторонніх рішень.	Монолітний фреймворк із широкими функціональними можливостями, що робить додатки більш важкими.
Частота використання	Vue.js молодший і лише набирає популярності. Однак на ринку вже існує багато сильних розробників Vue.js.	На даний момент React - це найпопулярніша технологія розробки інтерфейсів.	Angular - найстаріший фреймворк, використовуваний тисячами розробників у всьому світі. Набагато простіше знайти професіонала, який має кількарічний досвід.
Мова програмування	Чистий JavaScript	React використовує JSX - трохи змінену версію JavaScript. Це дозволяє більш ефективно програмувати, але потребує певного часу для ознайомлення.	TypeScript для кращої продуктивності або чистий JavaScript.
Крива навчання	Vue.js має найнижчу криву навчання серед усіх фронтальних рамок.	React складніше в навчанні, але введення React-hooks полегшує навчання.	Angular найскладніший у навчанні, проте легкий у освоєнні для розробників, що добре володіють ООП та мають досвід роботи з С-подібними мовами
Технічна документація та підтримка	Документація доступна. Спільнота розробників постійно зростає та розвивається.	Як найбільш поширена бібліотека для інтернету, React чудово склав документацію англійською мовою. Спільнота розмовляє різними мовами.	Відмінна детальна офіційна документація, десятки вичерпних навчальних посібників та надзвичайно прихильна спільнота. Відповідь на будь-яке питання можна знайти в Інтернеті.

ДОДАТОК В

Результати тестування системи за допомогою сервісу Google PageSpeed Insights



ДОДАТОК Г

Лістинг реалізації деяких модулів системи

app.module.ts

```
1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3.
4. import { AppRoutingModule } from './app-routing.module';
5. import { AppComponent } from './app.component';
6. import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
7. import { SharedModule } from './shared/shared.module';
8. import { AuthModule } from './auth/auth.module';
9. import { DashboardModule } from './dashboard/dashboard.module';
10. import { TestModule } from './test/test.module';
11.
12. @NgModule({
13.   declarations: [
14.     AppComponent,
15.   ],
16.   imports: [
17.     BrowserModule,
18.     SharedModule,
19.     AppRoutingModule,
20.     BrowserAnimationsModule,
21.     AuthModule,
22.     DashboardModule,
23.     SharedModule,
24.     TestModule,
25.     DashboardModule
26.   ],
27.   providers: [],
28.   bootstrap: [AppComponent]
29. })
30. export class AppModule { }
31.
```

shared.module.ts

```
1. import { CommonModule } from '@angular/common';
2. import { NgModule } from '@angular/core';
3. import { FormsModule, ReactiveFormsModule } from '@angular/forms';
4. import { HttpClientModule } from '@angular/common/http';
5. import { RouterModule } from '@angular/router';
6.
7. import { LayoutModule } from '@angular/cdk/layout';
8. import { MatCardModule } from '@angular/material/card';
9. import { MatToolbarModule } from '@angular/material/toolbar';
10. import { MatButtonModule } from '@angular/material/button';
11. import { MatListModule } from '@angular/material/list';
12. import { MatIconModule } from '@angular/material/icon';
13. import { MatSidenavModule } from '@angular/material/sidenav';
14. import { MatDialogModule } from '@angular/material/dialog';
15. import { MatExpansionModule } from '@angular/material/expansion';
16. import { MatMenuModule } from '@angular/material/menu';
17. import { MatGridListModule } from '@angular/material/grid-list';
18. import { MatTabsModule } from '@angular/material/tabs';
19. import { MatSelectModule } from '@angular/material/select';
20. import { MatInputModule } from '@angular/material/input';
21. import { MatCheckboxModule } from '@angular/material/checkbox';
22. import { MatRadioModule } from '@angular/material/radio';
23. import { MatDatepickerModule } from '@angular/material/datepicker';
24. import { MatNativeDateModule } from '@angular/material/core';
25. import { MatSlideToggleModule } from '@angular/material/slide-toggle';
26. import { MatProgressBarModule } from '@angular/material/progress-bar';
27. import { MatSnackBarModule } from '@angular/material/snack-bar';
28. import { MatTooltipModule } from '@angular/material/tooltip';
29. import { MatTableModule } from '@angular/material/table';
```

```
30. import { MatSortModule } from '@angular/material/sort';
31. import { MatPaginatorModule } from '@angular/material/paginator';
32.
33. import { PerfectScrollbarModule } from 'ngx-perfect-scrollbar';
34.
35. import {
36.   AccountCardComponent, AuthLayoutComponent,
37.   HeaderComponent,
38.   LogoutButtonComponent,
39.   MainLayoutComponent,
40.   MainNavComponent, MenuComponent, SidenavComponent,
41.   ToolbarComponent
42. } from './components/layout';
43. import { FooterComponent } from './components/layout';
44. import { ListErrorsComponent } from './components/errors';
45. import { LoginFormComponent } from './components/forms';
46. import { RegisterFormComponent } from './components/forms';
47. import {
48.   ButtonComponent,
49.   CheckboxComponent,
50.   DividerComponent,
51.   InputComponent,
52.   LinkComponent, LocalLinkComponent,
53.   OuterLinkComponent,
54.   PaginatorComponent, SectionHeaderComponent, SubheaderComponent, TextareaComponent
55. } from './components/UI';
56.
57. import { ValidationService } from './services';
58. import { ThemingModule } from '../theming/theming.module';
59. import { DialogLogoutComponent, DialogRemoveItemComponent } from './components/dialogs';
60. import { BreadcrumbComponent } from './components/breadcrumb/breadcrumb.component';
61. import { SpinnerComponent } from './components';
62. import { SelectComponent } from './components/UI/select/select.component';
63. import { MultiselectComponent } from './components/UI/multiselect/multiselect.component';
64.
65. @NgModule({
66.   imports: [
67.     CommonModule,
68.     FormsModule,
69.     ReactiveFormsModule,
70.     HttpClientModule,
71.     RouterModule,
72.     // Material
73.     LayoutModule,
74.     MatButtonModule,
75.     MatCardModule,
76.     MatCheckboxModule,
77.     MatDatepickerModule,
78.     MatDialogModule,
79.     MatExpansionModule,
80.     MatGridListModule,
81.     MatIconModule,
82.     MatInputModule,
83.     MatListModule,
84.     MatMenuModule,
85.     MatNativeDateModule,
86.     MatPaginatorModule,
87.     MatProgressBarModule,
88.     MatRadioModule,
89.     MatSelectModule,
90.     MatSidenavModule,
91.     MatSlideToggleModule,
92.     MatSnackBarModule,
93.     MatSortModule,
94.     MatTableModule,
95.     MatTabsModule,
96.     MatToolbarModule,
97.     MatTooltipModule,
98.     // End Material
99.     PerfectScrollbarModule,
100.     ThemingModule
101.   ],
102.   declarations: [
```

```
    HeaderComponent,
    FooterComponent,
    ListErrorsComponent,
    LoginFormComponent,
    RegisterFormComponent,
    ButtonComponent,
    CheckboxComponent,
    DividerComponent,
    InputComponent,
    LinkComponent,
    OuterLinkComponent,
    PaginatorComponent,
    SectionHeaderComponent,
    SubheaderComponent,
    TextareaComponent,
    LocalLinkComponent,
    ToolbarComponent,
    LogoutButtonComponent,
    AccountCardComponent,
    MainNavComponent,
    MainLayoutComponent,
    AuthLayoutComponent,
    DialogRemoveItemComponent,
    DialogLogoutComponent,
    BreadcrumbComponent,
    MenuComponent,
    SidenavComponent,
    SpinnerComponent,
    SelectComponent,
    MultiselectComponent
  ],
  providers: [
    ValidationService,
  ],
  exports: [
    CommonModule,
    FormsModule,
    ReactiveFormsModule,
    ReactiveFormsModule,
    HttpClientModule,
    RouterModule,
    HeaderComponent,
    FooterComponent,
    ListErrorsComponent,
    ButtonComponent,
    CheckboxComponent,
    DividerComponent,
    InputComponent,
    LinkComponent,
    OuterLinkComponent,
    PaginatorComponent,
    SectionHeaderComponent,
    SubheaderComponent,
    TextareaComponent,
    LoginFormComponent,
    RegisterFormComponent,
    LocalLinkComponent,
    ToolbarComponent,
    LogoutButtonComponent,
    AccountCardComponent,
    MainNavComponent,
    MainLayoutComponent,
    AuthLayoutComponent,
    DialogRemoveItemComponent,
    DialogLogoutComponent,
    BreadcrumbComponent,
    MenuComponent,
    SidenavComponent,
    SpinnerComponent,
    /**
     * Material
     */
    // Navigation
```

```
    MatMenuModule,
    MatSidenavModule,
    MatToolbarModule,
    // Layout
    MatListModule,
    MatCardModule,
    MatGridListModule,
    MatExpansionModule,
    MatTabsModule,
    // Form Controls
    MatInputModule,
    MatSelectModule,
    MatCheckboxModule,
    MatRadioModule,
    MatDatepickerModule,
    MatNativeDateModule,
    MatSlideToggleModule,
    // Buttons & indicators
    MatButtonModule,
    MatIconModule,
    MatProgressBarModule,
    // Popups / modals
    MatDialogModule,
    MatSnackBarModule,
    MatTooltipModule,
    // Data table
    MatTableModule,
    MatSortModule,
    MatPaginatorModule,
    /* End Material */
    PerfectScrollbarModule,
    ThemingModule
  ]
})
export class SharedModule {}
```

api.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { HttpTokenInterceptor } from './interceptors';
import {
  ApiService,
  JwtService,
  AuthApiService,
  TestApiService,
  TopicTestApiService,
  HelperService
} from './services';
import { LocalStorageModule } from '../local-storage/local-storage.module';
import { ToastModule } from '../toast';

@NgModule({
  declarations: [],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: HttpTokenInterceptor, multi: true },
    ApiService,
    JwtService,
    AuthApiService,
    TestApiService,
    TopicTestApiService,
    HelperService,
  ],
  imports: [
    CommonModule,
    HttpClientModule,
    LocalStorageModule,
    ToastModule
  ]
```

```
33. })
34. export class ApiModule { }
```

Лістинг реалізації деяких сервісів системи

api.service.ts

```
1. import { Injectable } from '@angular/core';
2. import { HttpClient, HttpParams } from '@angular/common/http';
3. import { throwError } from 'rxjs';
4. import { environment } from '../../../environments/environment';
5. import { catchError } from 'rxjs/operators';
6. import { JwtService } from './jwt.service';
7.
8. import { RequestResponse } from '../models/request-response';
9. import { HelperService } from './helper.service';
10. import { ToastService } from '../../toast';
11. import { ToastData } from '../../toast/toast-config';
12.
13. @Injectable()
14. export class ApiService {
15.   private SERVER_URL = environment.api_url;
16.
17.   constructor(
18.     private http: HttpClient,
19.     private jwtService: JwtService,
20.     private helperService: HelperService,
21.     private toastService: ToastService,
22.   ) { }
23.
24.   private errorInterceptor(error: any) {
25.     console.log(error);
26.     const data = new ToastData();
27.     data.text = error.message;
28.     data.title = error.statusText;
29.     data.type = 'warning';
30.     this.toastService.show(data);
31.     return throwError(error.error);
32.   }
33.
34.   private statusInterceptor<T>(response: RequestResponse<T>): T {
35.     if (response.Success === true) {
36.       return response.Data;
37.     }
38.     throwError(response.Message);
39.   }
40.
41.   async get<T>(path: string, params: HttpParams = new HttpParams()): Promise<T> {
42.     this.helperService.startLoader();
43.     const response = await this.http.get<RequestResponse<T>>(`${this.SERVER_URL}/${path}`, { params })
44.       .pipe(catchError(error => this.errorInterceptor(error)))
45.       .toPromise()
46.       .then((res) => {
47.         this.helperService.stopLoader();
48.         return res;
49.       });
50.
51.     return this.statusInterceptor<T>(response);
52.   }
53.
54.   async put<T>(path: string, body: any = {}): Promise<T> {
55.     this.helperService.startLoader();
56.     const response = await this.http.put<RequestResponse<T>>(`${this.SERVER_URL}/${path}`, JSON.stringify(body))
57.       .pipe(catchError(error => this.errorInterceptor(error)))
58.       .toPromise()
59.       .then((res) => {
60.         this.helperService.stopLoader();
61.         return res;
62.       });
63.
64.     return this.statusInterceptor<T>(response);
65.   }
```

```
66.

67.  async post<T>(path: string, body: any = {}): Promise<T> {
68.    this.helperService.startLoader();
69.    const response = await this.http.post<RequestResponse<T>>(`${this.SERVER_URL}/${path}`, JSON.stringify(body))
70.      .pipe(catchError(error => this.errorInterceptor(error)))
71.      .toPromise()
72.      .then((res) => {
73.        this.helperService.stopLoader();
74.        return res;
75.      });
76.
77.    return this.statusInterceptor<T>(response);
78.  }
79.
80.  async delete<T>(path): Promise<T> {
81.    this.helperService.startLoader();
82.    const response = await this.http.delete<RequestResponse<T>>(`${this.SERVER_URL}/${path}`)
83.      .pipe(catchError(error => this.errorInterceptor(error)))
84.      .toPromise()
85.      .then((res) => {
86.        this.helperService.stopLoader();
87.        return res;
88.      });
89.
90.    return this.statusInterceptor<T>(response);
91.  }
92. }
93.
```

validation.service.ts

```
1. import { Injectable } from '@angular/core';
2.
3. export enum ValidatorType {
4.    Email,
5.    Name,
6.    Comment,
7.    Phone,
8.    Password,
9.    Required,
10.    PassportNumber,
11.    Address,
12.    CompanyName,
13. }
14.
15. export enum ValidationType {
16.    OnSubmit,
17.    OnInput,
18. }
19.
20. interface IValidator {
21.    [key: string]: (value: string) => string[];
22. }
23.
24. type Validators = {
25.    [key: string]: IValidator;
26. }
27.
28. export const selectValidator = (
29.    validatorType: ValidatorType,
30.    validationType: ValidationType,
31. ): ((value: string) => string[]) | undefined => {
32.    switch (validatorType) {
33.       case ValidatorType.Email:
34.          switch (validationType) {
35.             case ValidationType.OnSubmit:
36.                return validators.Email.OnSubmit;
37.             case ValidationType.OnInput:
38.                return validators.Email.OnInput;
```

```
39.             default:
40.               return;

41.         }
42.       case ValidatorType.Name:
43.         switch (validationType) {
44.         case ValidationType.OnSubmit:
45.           return validators.Name.OnSubmit;
46.         case ValidationType.OnInput:
47.           return validators.Name.OnInput;
48.         default:
49.           return;
50.         }
51.       case ValidatorType.Phone:
52.         switch (validationType) {
53.         case ValidationType.OnSubmit:
54.           return validators.Phone.OnSubmit;
55.         case ValidationType.OnInput:
56.           return validators.Phone.OnInput;
57.         default:
58.           return;
59.         }
60.       case ValidatorType.Comment:
61.         switch (validationType) {
62.         case ValidationType.OnSubmit:
63.           return validators.Comment.OnSubmit;
64.         case ValidationType.OnInput:
65.           return validators.Comment.OnInput;
66.         default:
67.           return;
68.         }
69.       case ValidatorType.Password:
70.         switch (validationType) {
71.           case ValidationType.OnSubmit:
72.             return validators.Password.OnSubmit;
73.           case ValidationType.OnInput:
74.             return validators.Password.OnInput;
75.           default:
76.             return;
77.         }
78.       case ValidatorType.Required:
79.         switch (validationType) {
80.           case ValidationType.OnSubmit:
81.             return validators.Required.OnSubmit;
82.           case ValidationType.OnInput:
83.             return validators.Required.OnInput;
84.           default:
85.             return;
86.         }
87.       case ValidatorType.Address:
88.         switch (validationType) {
89.           case ValidationType.OnSubmit:
90.             return validators.Address.OnSubmit;
91.           case ValidationType.OnInput:
92.             return validators.Address.OnInput;
93.           default:
94.             return;
95.         }
96.       case ValidatorType.PassportNumber:
97.         switch (validationType) {
98.           case ValidationType.OnSubmit:
99.             return validators.PassportNumber.OnSubmit;
100.          case ValidationType.OnInput:
101.            return validators.PassportNumber.OnInput;
102.          default:
103.            return;
104.        }
105.     case ValidatorType.CompanyName:
106.       switch (validationType) {
107.         case ValidationType.OnSubmit:
108.           return validators.CompanyName.OnSubmit;
109.         case ValidationType.OnInput:
110.           return validators.CompanyName.OnInput;
```

```
            default:
              return;
          }

      default:
        return;
    }
};

export const validators: Validators = {
  Email: {
    OnSubmit: (value: string): string[] => {
      const errors: string[] = [];

      const validationPassed = value.match(
        /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/g,
      );

      if (!validationPassed) {
        errors.push('Unsupported format');
      }

      return errors;
    },
    OnInput: (value: string): string[] => {
      const errors: string[] = [];

      const validationPassed = value.match(/^[a-zA-Z0-9@_.+-]*$/);
      if (!validationPassed) {
        errors.push('Unsupported character');
      }

      return errors;
    },
  },
  Password: {
    OnSubmit: (value: string): string[] => {
      const errors: string[] = [];

      const minWidthPassed = value.length >= 8;
      if (!minWidthPassed) {
        errors.push('Password is too short');
      }

      const maxWidthPassed = value.length <= 30;
      if (!maxWidthPassed) {
        errors.push('Password is too long');
      }

      const validationPassed = value.match(
        /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$%^&+\\/'“?:.(){}[\]~\-_])[0-9a-zA-Z!@#$%^&+\\/'“?:.,(){}[\]~\-_]{8,}$/,
      );
      if (!validationPassed) {
        const hasSpecialCharacter = value.match(
          /[!@#$%^&+\\/'“?:.,(){}[\]~\-_]+/,
        );
        const hasNumber = value.match(/[0-9]+/);
        const hasUppercaseLetter = value.match(/[A-Z]+/);
        const hasLowercaseLetter = value.match(/[a-z]+/);
        if (!hasSpecialCharacter) {
          errors.push('Password must contain at least one special character');
        } else if (!hasNumber) {
          errors.push('Password must contain at least one number');
        } else if (!hasLowercaseLetter) {
          errors.push('Password must contain at least one lowercase letter');
        } else if (!hasUppercaseLetter) {
          errors.push('Password must contain at least one uppercase letter');
        } else {
          errors.push('Invalid password');
        }
      }
```

```
182.         return errors;
183.       },
184.       OnInput: (value: string): string[] => {

185.         const errors: string[] = [];
186.
187.         const validationPassed = value.match(
188.           /^[A-Za-z\d!@#$%^&+\\/'“?:.,(){}[\]~\-_\*]*$/,
189.         );
190.         if (!validationPassed) {
191.           errors.push('Unsupported character');
192.         }
193.
194.         return errors;
195.       },
196.     },
197.     Name: {
198.       OnSubmit: (value: string): string[] => {
199.         const errors: string[] = [];
200.
201.         const minWidthPassed = value.length >= 2;
202.         if (!minWidthPassed) {
203.           errors.push('Name is too short');
204.         }
205.
206.         const maxWidthPassed = value.length <= 30;
207.         if (!maxWidthPassed) {
208.           errors.push('Name is too long');
209.         }
210.
211.         const validationPassed = value.match(/^[a-zA-Z.-]*$/);
212.         if (!validationPassed) {
213.           errors.push('Unsupported character');
214.         }
215.
216.         return errors;
217.       },
218.       OnInput: (value: string): string[] => {
219.         const errors: string[] = [];
220.
221.         const validationPassed = value.match(/^[a-zA-Z.-]*$/);
222.         if (!validationPassed) {
223.           errors.push('Unsupported character');
224.         }
225.
226.         return errors;
227.       },
228.     },
229.     Comment: {
230.       OnSubmit: (value: string): string[] => {
231.         const errors: string[] = [];
232.
233.         const minWidthPassed = value.length >= 2;
234.         if (!minWidthPassed) {
235.           errors.push('Comment is too short');
236.         }
237.
238.         const maxWidthPassed = value.length <= 1000;
239.         if (!maxWidthPassed) {
240.           errors.push('Comment is too long');
241.         }
242.
243.         return errors;
244.       },
245.       OnInput: (value: string): string[] => {
246.         return [];
247.       },
248.     },
249.     Phone: {
250.       OnSubmit: (value: string): string[] => {
251.         const errors: string[] = [];
252.
```

```
253.          const validationPassed =
value.match(/\+(9[976]\d|8[987530]\d|6[987]\d|5[90]\d|42\d|3[875]\d|2[98654321]\d|9[8543210]|8[6421]|6[6543210]|5[87654321]|4[98
7654310]|3[9643210]|2[70]|7|1)\d{1,14}$/);
254.          if (!validationPassed) {
255.             errors.push('Unsupported format');

256.          }
257.
258.          return errors;
259.       },
260.       OnInput: (value: string): string[] => {
261.          const errors: string[] = [];
262.
263.          const validationPassed = value.match(/^[0-9+]*$/);
264.          if (!validationPassed) {
265.             errors.push('Unsupported character');
266.          }
267.
268.          return errors;
269.       },
270.    },
271.    Required: {
272.       OnSubmit: (value: string): string[] => {
273.          const errors: string[] = [];
274.
275.          const validationPassed = value.match(/^(?!\s*$).+/);
276.          if (!validationPassed) {
277.             errors.push('Field shouldn\'t be empty');
278.          }
279.
280.          return errors;
281.       },
282.       OnInput: (value: string): string[] => {
283.          return [];
284.       },
285.    },
286.    Address: {
287.       OnSubmit: (value: string): string[] => {
288.          return [];
289.       },
290.       OnInput: (value: string): string[] => {
291.          return [];
292.       },
293.    },
294.    PassportNumber: {
295.       OnSubmit: (value: string): string[] => {
296.          return [];
297.       },
298.       OnInput: (value: string): string[] => {
299.          return [];
300.       },
301.    },
302.    CompanyName: {
303.       OnSubmit: (value: string): string[] => {
304.          return [];
305.       },
306.       OnInput: (value: string): string[] => {
307.          return [];
308.       },
309.    },
310. };
311.
312. @Injectable()
313. export class ValidationService {
314.    public static validatorType = ValidatorType;
315.    public static validationType = ValidationType;
316.
317.    public validate(
318.       value: string,
319.       validatorTypes: ValidatorType[],
320.       validationType: ValidationType,
321.    ): string[] {
322.       return validatorTypes
```

```
323.          .reduce(
324.            (accumulator, currentValue) => {
325.              const validate = selectValidator(currentValue, validationType);
326.              if (validate) {
327.                const validationErrors = validate(value);
328.                return [...accumulator, ...validationErrors];

329.              }
330.              return accumulator;
331.            },
332.            [] as string[],
333.          )
334.          .filter(validationError => !!validationError);
335.    };
336. }
337.
```

local-storage.service.ts

```
1. import { Injectable } from '@angular/core';
2.
3. const MAX_STORE_TIME = 365 * 24 * 3600 * 1000;
4.
5. @Injectable()
6. export class LocalStorageService {
7.    getValue<T>(key: string, cacheKey?: string): T | false {
8.       const localKey = cacheKey ? key + cacheKey : key;
9.       try {
10.          const data = localStorage.getItem(localKey);
11.          const timestamp = this.getTimestamp(localKey);
12.          const maxStoreTime = this.getMaxStoreTime(localKey);
13.          if ((Date.now() - timestamp) > maxStoreTime) {
14.             return false;
15.          }
16.          return JSON.parse(data);
17.       } catch (e) {
18.          console.error(e);
19.          return false;
20.       }
21.    }
22.
23.    setValue<T>(value: T, key: string, cacheKey?: string, maxStoreTime?: number)
24.    {
25.       const localKey = cacheKey ? key + cacheKey : key;
26.       this.setTimestamp(localKey);
27.       this.setMaxStoreTime(localKey, maxStoreTime || MAX_STORE_TIME);
28.       const dataString = JSON.stringify(value);
29.       console.log(`setValue: ${localKey} <| ${dataString}`);
30.       localStorage.setItem(localKey, dataString);
31.    }
32.
33.    removeValue<T>(key: string, cacheKey?: string): boolean {
34.       const localKey = cacheKey ? key + cacheKey : key;
35.       try {
36.          const data = localStorage.removeItem(localKey);
37.          return true;
38.       } catch (e) {
39.          return false;
40.       }
41.    }
42.
43.    private getTimestamp(key: string): number {
44.       return +localStorage.getItem(`${key}-timestamp`);
45.    }
46.
47.    private getMaxStoreTime(key: string): number {
48.       return +localStorage.getItem(`${key}-maxStoreTime`);
49.    }
50.
51.    private setTimestamp(key: string) {
52.       const timestamp = Date.now();
```

```
53.         localStorage.setItem(`${key}-timestamp`, timestamp.toString());
54.     }
55.
56.     private setMaxStoreTime(key: string, timestamp: number) {
57.         localStorage.setItem(`${key}-maxStoreTime`, timestamp.toString());
58.     }
59. }
60.
```

Лістинг реалізації деяких шаблонів компонентів системи

register-form.component.html

```
<form class="RegisterForm">
  <div class="form-group">
   <app-input type="text"
          [placeholder]="'First Name'"
          name="register-first-name"
          [value]="firstName"
          [error]="firstNameError"
          (onValueChange)="handleFirstNameChange($event)"
          [componentSize]="'secondary'"></app-input>
  </div>
  <div class="form-group">
   <app-input type="text"
          [placeholder]="'Last Name'"
          name="register-last-name"
          [value]="lastName"
          [error]="lastNameError"
          (onValueChange)="handleLastNameChange($event)"
          [componentSize]="'secondary'"></app-input>
  </div>
  <div class="form-group">
   <app-input type="text"
          [placeholder]="'Email'"
          name="register-email"
          [value]="email"
          [error]="emailError"
          (onValueChange)="handleEmailChange($event)"
          [componentSize]="'secondary'"></app-input>
  </div>
  <div class="form-group">
   <app-input type="text"
          [placeholder]="'Password'"
          name="register-password"
          [value]="password"
          [error]="passwordError"
          (onValueChange)="handlePasswordChange($event)"
          [componentSize]="'secondary'"></app-input>
  </div>
  <section class="submit">
   <div class="d-flex justify-content-between align-items-center">
    <div [style.visibility]="waiting ? 'visible' : 'hidden'" class="spin-container animation-spin">
     <mat-icon aria-label="loading" color="primary" class="x2">autorenew</mat-icon>
    </div>
    <button (click)="submit()" mat-flat-button type="submit" [disabled]="waiting" color="primary">Login in</button>
   </div>
  </section>
</form>

```

input.component.html

```
<mat-form-field class="Input__container">
  <mat-icon *ngIf="icon" matPrefix>{{icon}}</mat-icon>
  <input
   [ngClass]="['Input', componentSizeClass, componentStyleClass]"
   matInput
   [name]="name"
   [type]="type"
   [placeholder]="placeholder"
   [disabled]="disabled"
    [value]="value"
   [formControl]="inputFormControl"
   [errorStateMatcher]="matcher"
   (change)="handleValueChange($event)"
   (input)="handleValueChange($event)"
  >
  <mat-error class="Error__container" *ngIf="error">
```

```
17.    <span>{{error}}</span>
18.  </mat-error>
19. </mat-form-field>
20.
```

auth-layout.component.html

```
1. <div id="Auth__wrapper">
2.    <app-header></app-header>
3.
4.    <div class="Auth">
5.       <div class="Auth__card__wrapper">
6.          <div class="Auth__card">
7.             <router-outlet>
8.                <!-- Add Content Here -->
9.                <ng-content></ng-content>
10.            </router-outlet>
11.         </div>
12.      </div>
13.   </div>
14.
15.   <!-- footer -->
16.   <app-footer></app-footer>
17. </div>
18.
```

Лістинг реалізації деяких каскадних таблиць стилів компонентів системи

auth-layout.component.scss

```
1. .Auth {
2.   width: 100%;
3.   height: calc(100vh - 128px);
4.   z-index: 10;
5.   background: linear-gradient(125deg, #e36f58 6%, #85b7c0 50%, #85b7c0 36%, #e5ab26 94%);
6.
7.   &__card {
8.     width: 35%;
9.     min-width: 340px;
10.    max-width: 580px;
11.    padding: 1.75% 2.5%;
12.    border: 0 solid #3f4e57;
13.    box-shadow: 0 0 10px -1px #3F4E5B;
14.    background-color: #fff;
15.    border-radius: 0.25rem;
16.
17.    &__wrapper {
18.      height: 100%;
19.      display: flex;
20.      justify-content: center;
21.      align-items: center;
22.    }
23.
24.    h2 {
25.      margin: 0 0 6% 0;
26.      font-size: 1.75rem;
27.      font-weight: 400;
28.    }
29.
30.    p {
31.      font-weight: 200;
32.    }
33.
34.    section {
35.      margin-top: 3%;
36.
37.      &.fields {
38.        margin: 3% 4% 5%;
39.
40.        &>div {
41.          margin-bottom: 8px;
42.        }
43.      }
44.
45.      &.submit {
46.        margin-top: 6%;
47.      }
48.
49.      .spin-container {
50.        width: 32px;
51.        height: 32px;
52.        margin-right: 12px;
53.      }
54.
55.      mat-form-field {
56.        width: 96%;
57.      }
58.    }
59.  }
60. }
61.
```

app.scss

```
1. // Material
2. @import '~@angular/material/theming';
3. // Plus imports for other components in your app.
```

```
4.
5. // Include the common styles for Angular Material. We include this here so that you only
6. // have to load a single css file for Angular Material in your app.
7. // Be sure that you only ever include this mixin once!
8. @include mat-core();
9.
10. // Import material icons
11. // from: https://jossef.github.io/material-design-icons-iconfont/
12. @import 'material-design-icons-iconfont/dist/material-design-icons.css';
13.
14. // Import material styles
15. @import '@material/layout-grid/dist/mdc.layout-grid.min.css';
16.
17. /**
18.  * START themes
19.  */
20.
21. // import default theme
22. @import './themes/default.scss';
23. @import './themes/minimal-light.scss';
24. @import './themes/minimal-dark.scss';
25.
26. // Override typography CSS classes (e.g., mat-h1, mat-display-1, mat-typography, etc.).
27. @include mat-base-typography($minimal-custom-typography);
28. // Override typography for a specific Angular Material components.
29. @include mat-checkbox-typography($minimal-custom-typography);
30. // Override typography for all Angular Material, including mat-base-typography and all components.
31. @include angular-material-typography($minimal-custom-typography);
32.
33. // Include a default theme at start
34. @include angular-material-theme($default-theme);
35.
36. .default-theme {
37.     @include angular-material-theme($default-theme);
38. }
39.
40. .minimal-light-theme {
41.     @include angular-material-theme($minimal-light-theme);
42. }
43.
44. .minimal-dark-theme {
45.     @include angular-material-theme($minimal-dark-theme);
46. }
47.
48. // END themes
49.
50. // Import beauty scrollbar
51. @import '~perfect-scrollbar/css/perfect-scrollbar.css';
52.
53. // Import Lato Font
54. @import './fonts.scss';
55.
56. // Own styles
57. @import './main.scss';
58. @import './spinner.scss';
59. @import './responsive.scss';
60.
```

spinner.scss

```
1. .preloader {
2.     position: absolute;
3.     margin: 0 auto;
4.     width: 100%;
5.     height: 100%;
6. }
7.
8. .spinner {
9.     width: 40px;
10.     height: 40px;
11.     top: 35%;
```

```
12.     position: relative;
13.     margin: 100px auto;
14. }
15.
16. .double-bounce1,
17. .double-bounce2 {
18.     width: 100%;
19.     height: 100%;
20.     border-radius: 50%;
21.     background-color: #1976d2;
22.     opacity: 0.6;
23.     position: absolute;
24.     top: 0;
25.     left: 0;
26.     -webkit-animation: sk-bounce 2.0s infinite ease-in-out;
27.     animation: sk-bounce 2.0s infinite ease-in-out;
28. }
29.
30. .double-bounce2 {
31.     -webkit-animation-delay: -1.0s;
32.     animation-delay: -1.0s;
33. }
34.
35. /*Preloader*/
36. .preloader {
37.     width: 100%;
38.     height: 100%;
39.     top: 0px;
40.     position: fixed;
41.     z-index: 99999;
42.     background: #fff;
43.
44.     .cssload-speeding-wheel {
45.         position: absolute;
46.         top: calc(50 % - 3.5px);
47.         left: calc(50 % - 3.5px);
48.     }
49. }
50.
51. @-webkit-keyframes sk-bounce {
52.
53.     0%,
54.     100% {
55.         -webkit-transform: scale(0.0)
56.     }
57.
58.     50% {
59.         -webkit-transform: scale(1.0)
60.     }
61. }
62.
63. @keyframes sk-bounce {
64.
65.     0%,
66.     100% {
67.         transform: scale(0.0);
68.         -webkit-transform: scale(0.0);
69.     }
70.
71.     50% {
72.         transform: scale(1.0);
73.         -webkit-transform: scale(1.0);
74.     }
75. }
76.
77. .spin {
78.     -webkit-animation: spin 1.33s linear infinite;
79.     -moz-animation: spin 1.33s linear infinite;
80.     animation: spin 1.33s linear infinite;
81.     padding: 6px 0;
82. }
83.
84. @-moz-keyframes spin {
```

```
85.     100% {

86.         -moz-transform: rotate(360deg);
87.     }
88. }
89.
90. @-webkit-keyframes spin {
91.     100% {
92.         -webkit-transform: rotate(360deg);
93.     }
94. }
95.
96. @keyframes spin {
97.     100% {
98.         -webkit-transform: rotate(360deg);
99.         transform: rotate(360deg);
100.    }
101. }
102.
```

Лістинг коду реалізації деяких компонентів

register-form.component.ts

```
1. import {Component, EventEmitter, Input, Output} from '@angular/core';
2. import { AuthService } from '../../../../auth/services';
3. import { ValidationService, ValidationType, ValidatorType } from '../../../services/validation.service';
4. import { RegisterDto } from '../../../../api/models/dto/auth';
5.
6. @Component({
7.   selector: 'app-register-form',
8.   templateUrl: './register-form.component.html',
9.   styleUrls: ['./register-form.component.scss']
10. })
11. export class RegisterFormComponent {
12.   @Input() waiting: boolean = false;
13.   @Output() onSubmit = new EventEmitter<RegisterDto>();
14.
15.   firstName: string = '';
16.   firstNameError: string = '';
17.   lastName: string = '';
18.   lastNameError: string = '';
19.   email: string = '';
20.   emailError: string = '';
21.   password: string = '';
22.   passwordError: string = '';
23.
24.   constructor(
25.     private readonly authService: AuthService,
26.     private readonly validationService: ValidationService,
27.   ) { }
28.
29.   handleFirstNameChange(value: string) {
30.     const validationErrors = this.validationService.validate(
31.       value,
32.       [ValidatorType.Name],
33.       ValidationType.OnInput,
34.     );
35.     this.firstNameError = validationErrors[0];
36.     this.firstName = value;
37.   }
38.
39.   handleLastNameChange(value: string) {
40.     const validationErrors = this.validationService.validate(
41.       value,
42.       [ValidatorType.Name],
43.       ValidationType.OnInput,
44.     );
45.     this.lastNameError = validationErrors[0];
46.     this.lastName = value;
47.   }
48.
49.   handleEmailChange(value: string) {
50.     const validationErrors = this.validationService.validate(
51.       value,
52.       [ValidatorType.Email],
53.       ValidationType.OnInput,
54.     );
55.     this.emailError = validationErrors[0];
56.     this.email = value;
57.   }
58.
59.   handlePasswordChange(value: string) {
60.     const validationErrors = this.validationService.validate(
61.       value,
62.       [ValidatorType.Password],
63.       ValidationType.OnInput,
64.     );
65.     this.passwordError = validationErrors[0];
66.     this.password = value;
67.   }
68.
```

```
69.  async submit() {
70.    const firstNameValidationErrors = this.validationService.validate(
71.      this.firstName,
72.      [ValidatorType.Required, ValidatorType.Name],
73.      ValidationType.OnSubmit,
74.    );
75.
76.    const lastNameValidationErrors = this.validationService.validate(
77.      this.lastName,
78.      [ValidatorType.Required, ValidatorType.Name],
79.      ValidationType.OnSubmit,
80.    );
81.
82.    const emailValidationErrors = this.validationService.validate(
83.      this.email,
84.      [ValidatorType.Required, ValidatorType.Email],
85.      ValidationType.OnSubmit,
86.    );
87.
88.    const passwordValidationErrors = this.validationService.validate(
89.      this.password,
90.      [ValidatorType.Required],
91.      ValidationType.OnSubmit,
92.    );
93.
94.    this.firstNameError = firstNameValidationErrors[0];
95.    this.lastNameError = lastNameValidationErrors[0];
96.    this.emailError = emailValidationErrors[0];
97.    this.passwordError = passwordValidationErrors[0];
98.
99.    const hasErrors = firstNameValidationErrors.length
100.       || lastNameValidationErrors.length
101.       || emailValidationErrors.length
102.       || passwordValidationErrors.length;
103.
104.     if (!hasErrors) {
105.       const data: RegisterDto = {
106.         Email: this.email,
107.         FirstName: this.firstName,
108.         LastName: this.lastName,
109.         Password: this.password
110.       };
111.
112.       this.waiting = true;
113.
114.       console.log(data);
115.
116.       this.onSubmit.emit(data);
117.     }
118.   }
119. }
120.
```

main-nav.component.ts

```
1. import {Component, ViewChild, EventEmitter, AfterViewInit} from '@angular/core';
2. import { BreakpointObserver, Breakpoints } from '@angular/cdk/layout';
3. import { Observable } from 'rxjs';
4. import { map } from 'rxjs/operators';
5.
6. import { MatSidenav } from '@angular/material/sidenav';
7. import { SETTINGS } from '../../../../models/UI/settings.model';
8.
9. @Component({
10.    selector: 'app-main-nav',
11.    templateUrl: './main-nav.component.html'
12. })
13. export class MainNavComponent implements AfterViewInit {
14.    public isOpened: boolean;
15.    public hasMiniSidenav: boolean;
16.    public hasToggleMiniSidenav: boolean;
```

```
17.   public toggleSideMenu: boolean;
18.   public toggleSideMenu$: EventEmitter<boolean>;
19.
20.   @ViewChild('drawer') sideNav: MatSidenav;
21.
22.   isHandset$: Observable<boolean> = this.breakpointObserver.observe(Breakpoints.Handset)
23.     .pipe(
24.       map(result => result.matches)
25.     );
26.
27.   constructor(private breakpointObserver: BreakpointObserver) {
28.     this.isOpened = false;
29.     this.toggleSideMenu = false;
30.     this.toggleSideMenu$ = new EventEmitter(this.toggleSideMenu);
31.     this.hasMiniSidenav = SETTINGS.hasMiniSidenav;
32.     this.hasToggleMiniSidenav = SETTINGS.hasMiniSidenav;
33.   }
34.
35.   ngAfterViewInit() {
36.    this.sideNav.openedStart.subscribe(
37.     (() => this.isOpened = true)
38.    );
39.
40.    this.sideNav.openedChange.subscribe(
41.     (res => this.isOpened = res)
42.    );
43.
44.    this.toggleSideMenu$.subscribe(
45.     (res => this.toggleSideMenu = res)
46.    );
47.   }
48. }
49.
```

toast.component.ts

```
1. import { Component, OnInit, OnDestroy, Inject } from '@angular/core';
2. import { AnimationEvent } from '@angular/animations';
3.
4. import { ToastData, TOAST_CONFIG_TOKEN, ToastConfig } from './toast-config';
5. import { ToastRef } from './toast-ref';
6. import { toastAnimations, ToastAnimationState } from './toast-animation';
7.
8. @Component({
9.   selector: 'app-toast',
10.   templateUrl: './toast.component.html',
11.   styleUrls: ['toast.component.scss'],
12.   animations: [toastAnimations.fadeToast],
13. })
14. export class ToastComponent implements OnInit, OnDestroy {
15.   animationState: ToastAnimationState = 'default';
16.   iconType: string;
17.
18.   private intervalId;
19.
20.   constructor(
21.     readonly data: ToastData,
22.     readonly ref: ToastRef,
23.     @Inject(TOAST_CONFIG_TOKEN) public toastConfig: ToastConfig
24.   ) {
25.     this.iconType = data.type === 'success' ? 'done' : data.type;
26.   }
27.
28.   ngOnInit() {
29.     this.intervalId = setTimeout(() => this.animationState = 'closing', 5000);
30.   }
31.
32.   ngOnDestroy() {
33.     clearTimeout(this.intervalId);
34.   }
35.
36.   close() {
```

```
37.       this.ref.close();
38.     }
39.
40.     onFadeFinished(event: AnimationEvent) {
41.       const { toState } = event;
42.       const isFadeOut = (toState as ToastAnimationState) === 'closing';
43.       const itFinished = this.animationState === 'closing';
44.
45.       if (isFadeOut && itFinished) {
46.         this.close();
47.       }
48.     }
49. }
50.
```

ДОДАТОК Д

Структурна організація коду в межах файлової системи ОС

```
|   .editorconfig
|   .gitignore
|   angular.json
|   browserslist
|   karma.conf.js
|   output.doc
|   package-lock.json
|   package.json
|   README.md
|   tsconfig.app.json
|   tsconfig.json
|   tsconfig.spec.json
|   tslint.json
|
+---.idea
|   |   misc.xml
|   |   modules.xml
|   |   testing-system.iml
|   |   vcs.xml
|   |   workspace.xml
|   |
|   +---codeStyles
|   |       codeStyleConfig.xml
|   |       Project.xml
|   |
|   \---inspectionProfiles
|           Project_Default.xml
|
\---src
    |   favicon.ico
    |   index.html
    |   main.ts
    |   polyfills.ts
    |   test.ts
    |
    +---app
    |   |   app-routing.module.ts
    |   |   app.component.html
    |   |   app.component.spec.ts
    |   |   app.component.ts
    |   |   app.module.ts
    |   |
    |   +---api
    |   |   |   api.module.ts
    |   |   |
    |   |   +---interceptors
    |   |   |       http.token.interceptor.ts
    |   |   |       index.ts
    |   |   |
    |   |   +---models
    |   |   |   |   request-response.ts
    |   |   |   |
    |   |   |   +---dto
    |   |   |   |   |   index.ts
    |   |   |   |   |
    |   |   |   |   +---auth
    |   |   |   |   |       index.ts
    |   |   |   |   |       login.dto.ts
    |   |   |   |   |       refresh.dto.ts
    |   |   |   |   |       register.dto.ts
    |   |   |   |   |
    |   |   |   |   +---test
    |   |   |   |   |       add-question.dto.ts
    |   |   |   |   |       add-test.dto.ts
    |   |   |   |   |       check-question.dto.ts
    |   |   |   |   |       index.ts
    |   |   |   |   |
    |   |   |   |   \---test-topic
    |   |   |   |           index.ts
    |   |   |   |
```

```
|   |   |   \---ro
|   |   |       |   index.ts
|   |   |       |
|   |   |       +---auth
|   |   |       |       index.ts
|   |   |       |       login.ro.ts
|   |   |       |       refresh.ro.ts
|   |   |       |       register.ro.ts
|   |   |       |
|   |   |       +---test
|   |   |       |       add-question.ro.ts
|   |   |       |       add-test.ro.ts
|   |   |       |       check-question.ro.ts
|   |   |       |       get-test.ro.ts
|   |   |       |       get-tests.ro.ts
|   |   |       |       index.ts
|   |   |       |
|   |   |       \---topic-test
|   |   |               get-test-topic.ro.ts
|   |   |               index.ts
|   |   |
|   |   \---services
|   |       |   api.service.spec.ts
|   |       |   api.service.ts
|   |       |   helper.service.ts
|   |       |   index.ts
|   |       |   jwt.service.ts
|   |       |
|   |       \---data
|   |               auth-api.service.ts
|   |               auth.service.spec.ts
|   |               index.ts
|   |               test-api.service.ts
|   |               test.service.spec.ts
|   |               topic-test-api.service.ts
|   |               topic-test.service.spec.ts
|   |
|   +---auth
|   |   |   auth-routing.module.ts
|   |   |   auth.module.ts
|   |   |
|   |   +---components
|   |   |   |   index.ts
|   |   |   |
|   |   |   +---login
|   |   |   |       login.component.html
|   |   |   |       login.component.scss
|   |   |   |       login.component.ts
|   |   |   |
|   |   |   \---register
|   |   |           register.component.html
|   |   |           register.component.scss
|   |   |           register.component.ts
|   |   |
|   |   +---directives
|   |   |       index.ts
|   |   |       show-authed.directive.ts
|   |   |
|   |   +---guargs
|   |   |       auth-guard.service.ts
|   |   |       index.ts
|   |   |       no-auth-guard.service.ts
|   |   |
|   |   \---services
|   |           auth.service.ts
|   |           index.ts
|   |
|   +---dashboard
|   |   |   dashboard-routing.module.ts
|   |   |   dashboard.module.ts
|   |   |
|   |   +---components
|   |   |   |   index.ts
|   |   |   |
|   |   |   +---dashboard
```

```
|   |   |   |       dashboard.component.html
|   |   |   |       dashboard.component.scss
|   |   |   |       dashboard.component.spec.ts
|   |   |   |       dashboard.component.ts
|   |   |   |
|   |   |   \---history
|   |   |           history.component.html
|   |   |           history.component.scss
|   |   |           history.component.spec.ts
|   |   |           history.component.ts
|   |   |
|   |   \---services
|   |           dashboard.service.spec.ts
|   |           dashboard.service.ts
|   |           index.ts
|   |
|   +---local-storage
|   |       local-storage.module.ts
|   |       local-storage.service.ts
|   |
|   +---models
|   |   |   index.ts
|   |   |
|   |   +---data
|   |   |       index.ts
|   |   |       question.interface.ts
|   |   |       test-data.interface.ts
|   |   |       user.interface.ts
|   |   |       version.interface.ts
|   |   |
|   |   \---UI
|   |           menu.model.ts
|   |           settings.model.ts
|   |
|   +---settings
|   |       app.config.ts
|   |       index.ts
|   |
|   +---shared
|   |   |   shared.module.ts
|   |   |
|   |   +---components
|   |   |   |   index.ts
|   |   |   |
|   |   |   +---breadcrumb
|   |   |   |       breadcrumb.component.html
|   |   |   |       breadcrumb.component.spec.ts
|   |   |   |       breadcrumb.component.ts
|   |   |   |       breadcrumb.interface.ts
|   |   |   |
|   |   |   +---dialogs
|   |   |   |   |   index.ts
|   |   |   |   |
|   |   |   |   +---dialog-logout
|   |   |   |   |       dialog-logout.component.html
|   |   |   |   |       dialog-logout.component.spec.ts
|   |   |   |   |       dialog-logout.component.ts
|   |   |   |   |
|   |   |   |   \---dialog-remove-item
|   |   |   |           dialog-remove-item.component.html
|   |   |   |           dialog-remove-item.component.spec.ts
|   |   |   |           dialog-remove-item.component.ts
|   |   |   |
|   |   |   +---errors
|   |   |   |   |   index.ts
|   |   |   |   |
|   |   |   |   \---list-errors
|   |   |   |           list-errors.component.html
|   |   |   |           list-errors.component.ts
|   |   |   |
|   |   |   +---forms
|   |   |   |   |   index.ts
|   |   |   |   |
|   |   |   |   +---login-form
|   |   |   |   |       login-form.component.html
```

```
|   |   |   |   |       login-form.component.scss
|   |   |   |   |       login-form.component.spec.ts
|   |   |   |   |       login-form.component.ts
|   |   |   |   |
|   |   |   |   \---register-form
|   |   |   |           register-form.component.html
|   |   |   |           register-form.component.scss
|   |   |   |           register-form.component.spec.ts
|   |   |   |           register-form.component.ts
|   |   |   |
|   |   |   +---layout
|   |   |   |   |   index.ts
|   |   |   |   |
|   |   |   |   +---auth-layout
|   |   |   |   |       auth-layout.component.html
|   |   |   |   |       auth-layout.component.scss
|   |   |   |   |       auth-layout.component.spec.ts
|   |   |   |   |       auth-layout.component.ts
|   |   |   |   |
|   |   |   |   +---footer
|   |   |   |   |       footer.component.html
|   |   |   |   |       footer.component.scss
|   |   |   |   |       footer.component.spec.ts
|   |   |   |   |       footer.component.ts
|   |   |   |   |
|   |   |   |   +---header
|   |   |   |   |       header.component.html
|   |   |   |   |       header.component.scss
|   |   |   |   |       header.component.spec.ts
|   |   |   |   |       header.component.ts
|   |   |   |   |
|   |   |   |   +---main-layout
|   |   |   |   |       main-layout.component.html
|   |   |   |   |       main-layout.component.spec.ts
|   |   |   |   |       main-layout.component.ts
|   |   |   |   |
|   |   |   |   +---main-nav
|   |   |   |   |       main-nav.component.html
|   |   |   |   |       main-nav.component.spec.ts
|   |   |   |   |       main-nav.component.ts
|   |   |   |   |
|   |   |   |   +---sidenav
|   |   |   |   |   |   index.ts
|   |   |   |   |   |   sidenav.component.html
|   |   |   |   |   |   sidenav.component.spec.ts
|   |   |   |   |   |   sidenav.component.ts
|   |   |   |   |   |
|   |   |   |   |   \---menu
|   |   |   |   |           menu.component.html
|   |   |   |   |           menu.component.spec.ts
|   |   |   |   |           menu.component.ts
|   |   |   |   |
|   |   |   |   \---toolbar
|   |   |   |       |   index.ts
|   |   |   |       |   toolbar.component.html
|   |   |   |       |   toolbar.component.spec.ts
|   |   |   |       |   toolbar.component.ts
|   |   |   |       |
|   |   |   |       +---account-card
|   |   |   |       |       account-card.component.html
|   |   |   |       |       account-card.component.scss
|   |   |   |       |       account-card.component.spec.ts
|   |   |   |       |       account-card.component.ts
|   |   |   |       |
|   |   |   |       \---logout-button
|   |   |   |               logout-button.component.html
|   |   |   |               logout-button.component.spec.ts
|   |   |   |               logout-button.component.ts
|   |   |   |
|   |   |   +---spinner
|   |   |   |       spinner.component.html
|   |   |   |       spinner.component.spec.ts
|   |   |   |       spinner.component.ts
|   |   |   |
|   |   |   \---UI
```

```
|   |   |       |   index.ts
|   |   |       |
|   |   |       +---button
|   |   |       |       button.component.html
|   |   |       |       button.component.scss
|   |   |       |       button.component.ts
|   |   |       |
|   |   |       +---checkbox
|   |   |       |       checkbox.component.html
|   |   |       |       checkbox.component.scss
|   |   |       |       checkbox.component.ts
|   |   |       |
|   |   |       +---divider
|   |   |       |       divider.component.html
|   |   |       |       divider.component.scss
|   |   |       |       divider.component.ts
|   |   |       |
|   |   |       +---input
|   |   |       |       input.component.html
|   |   |       |       input.component.scss
|   |   |       |       input.component.ts
|   |   |       |
|   |   |       +---link
|   |   |       |       link.component.html
|   |   |       |       link.component.scss
|   |   |       |       link.component.ts
|   |   |       |
|   |   |       +---local-link
|   |   |       |       local-link.component.html
|   |   |       |       local-link.component.scss
|   |   |       |       local-link.component.ts
|   |   |       |
|   |   |       +---multiselect
|   |   |       |       multiselect.component.html
|   |   |       |       multiselect.component.scss
|   |   |       |       multiselect.component.spec.ts
|   |   |       |       multiselect.component.ts
|   |   |       |
|   |   |       +---outer-link
|   |   |       |       outer-link.component.html
|   |   |       |       outer-link.component.scss
|   |   |       |       outer-link.component.ts
|   |   |       |
|   |   |       +---paginator
|   |   |       |       paginator.component.html
|   |   |       |       paginator.component.scss
|   |   |       |       paginator.component.ts
|   |   |       |
|   |   |       +---section-header
|   |   |       |       section-header.component.html
|   |   |       |       section-header.component.scss
|   |   |       |       section-header.component.ts
|   |   |       |
|   |   |       +---select
|   |   |       |       select.component.html
|   |   |       |       select.component.scss
|   |   |       |       select.component.spec.ts
|   |   |       |       select.component.ts
|   |   |       |
|   |   |       +---subheader
|   |   |       |       subheader.component.html
|   |   |       |       subheader.component.scss
|   |   |       |       subheader.component.ts
|   |   |       |
|   |   |       \---textarea
|   |   |               textarea.component.html
|   |   |               textarea.component.scss
|   |   |               textarea.component.ts
|   |   |
|   |   \---services
|   |           index.ts
|   |           validation.service.ts
|   |
|   +---test
|   |   |   test.module.ts
```

```
|   |   |
|   |   +---components
|   |   |   |   index.ts
|   |   |   |
|   |   |   +---question
|   |   |   |       question.component.html
|   |   |   |       question.component.scss
|   |   |   |       question.component.spec.ts
|   |   |   |       question.component.ts
|   |   |   |
|   |   |   +---question-type-selector
|   |   |   |       question-type-selector.component.html
|   |   |   |       question-type-selector.component.scss
|   |   |   |       question-type-selector.component.spec.ts
|   |   |   |       question-type-selector.component.ts
|   |   |   |
|   |   |   +---questions
|   |   |   |   |   index.ts
|   |   |   |   |
|   |   |   |   +---question-compilance
|   |   |   |   |       question-compilance.component.html
|   |   |   |   |       question-compilance.component.scss
|   |   |   |   |       question-compilance.component.spec.ts
|   |   |   |   |       question-compilance.component.ts
|   |   |   |   |
|   |   |   |   +---question-multiselect
|   |   |   |   |       question-multiselect.component.html
|   |   |   |   |       question-multiselect.component.scss
|   |   |   |   |       question-multiselect.component.spec.ts
|   |   |   |   |       question-multiselect.component.ts
|   |   |   |   |
|   |   |   |   +---question-number
|   |   |   |   |       question-number.component.html
|   |   |   |   |       question-number.component.scss
|   |   |   |   |       question-number.component.spec.ts
|   |   |   |   |       question-number.component.ts
|   |   |   |   |
|   |   |   |   +---question-select
|   |   |   |   |       question-select.component.html
|   |   |   |   |       question-select.component.scss
|   |   |   |   |       question-select.component.spec.ts
|   |   |   |   |       question-select.component.ts
|   |   |   |   |
|   |   |   |   +---question-sequence
|   |   |   |   |       question-sequence.component.html
|   |   |   |   |       question-sequence.component.scss
|   |   |   |   |       question-sequence.component.spec.ts
|   |   |   |   |       question-sequence.component.ts
|   |   |   |   |
|   |   |   |   +---question-short
|   |   |   |   |       question-short.component.html
|   |   |   |   |       question-short.component.scss
|   |   |   |   |       question-short.component.spec.ts
|   |   |   |   |       question-short.component.ts
|   |   |   |   |
|   |   |   |   \---question-truthfulness
|   |   |   |           question-truthfulness.component.html
|   |   |   |           question-truthfulness.component.scss
|   |   |   |           question-truthfulness.component.spec.ts
|   |   |   |           question-truthfulness.component.ts
|   |   |   |
|   |   |   +---reference
|   |   |   |       reference.component.html
|   |   |   |       reference.component.scss
|   |   |   |       reference.component.spec.ts
|   |   |   |       reference.component.ts
|   |   |   |
|   |   |   +---test
|   |   |   |       test.component.html
|   |   |   |       test.component.scss
|   |   |   |       test.component.spec.ts
|   |   |   |       test.component.ts
|   |   |   |
|   |   |   +---test-list
|   |   |   |       test-list.component.html
```

```
|   |   |   |       test-list.component.scss
|   |   |   |       test-list.component.spec.ts
|   |   |   |       test-list.component.ts
|   |   |   |
|   |   |   \---test-step
|   |   |           test-step.component.html
|   |   |           test-step.component.scss
|   |   |           test-step.component.spec.ts
|   |   |           test-step.component.ts
|   |   |
|   |   \---services
|   |           index.ts
|   |           test-controller.service.spec.ts
|   |           test-controller.service.ts
|   |           test.service.spec.ts
|   |           test.service.ts
|   |
|   +---theming
|   |       theming.module.ts
|   |       theming.service.spec.ts
|   |       theming.service.ts
|   |
|   \---toast
|           index.ts
|           toast-animation.ts
|           toast-config.ts
|           toast-ref.ts
|           toast.component.html
|           toast.component.scss
|           toast.component.ts
|           toast.module.ts
|           toast.service.ts
|
+---assets
|   |   .gitkeep
|   |
|   +---fonts
|   |       lato_bold-italic.woff2
|   |       lato_bold.woff2
|   |       lato_italic.woff2
|   |       lato_light.woff2
|   |       lato_regular.woff2
|   |
|   +---images
|   |       logo-icon.png
|   |       logo.png
|   |       logo_alt.png
|   |       logo_alt2.png
|   |
|   \---scss
|       |   app.scss
|       |   colors.scss
|       |   fonts.scss
|       |   main.scss
|       |   responsive.scss
|       |   spinner.scss
|       |
|       \---themes
|               default.scss
|               minimal-dark.scss
|               minimal-light.scss
|
\---environments
        environment.prod.ts
        environment.ts
```